Εθνικο Μετσοβιο Πολυτεχνειο
Τμημα Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Σηματων Ελεγχου και Ρομποτικης

Ημιεπιβλεπόμενη μάθηση για την ανάλυση ηχογραφήσεων ποντικιών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασιλική Στούμπου

Εξωτερικός Επιβλέπων: Θεόδωρος Γιαννακόπουλος
Β΄ Ερευνητής ΕΚΕΦΕ Δημόκριτος

Επιβλέπων Ε.Μ.Π.: Αλέξανδρος Ποταμιάνος
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2021

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Ημιεπιβλεπόμενη μάθηση για την ανάλυση ηχογραφήσεων ποντικιών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασιλική Στούμπου

Εξωτερικός Επιβλέπων: Θεόδωρος Γιαννακόπουλος
Β΄ Ερευνητής ΕΚΕΦΕ Δημόκριτος

Επιβλέπων Ε.Μ.Π.: Αλέξανδρος Ποταμιάνος
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 5η Ιουλίου 2021.

.....	.....	.....
Αλέξανδρος Ποταμιάνος Αναπληρωτής Καθηγητής Ε.Μ.Π.	Θεόδωρος Γιαννακόπουλος Β΄ Ερευνητής ΕΚΕΦΕ Δημόκριτος	Στέφανος Κόλλιας Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2021

National Technical University Of Athens
School of Electrical And Computer Engineering
Signals of Control And Robotics

Semi-Supervised learning for mice recordings analysis

DIPLOMA THESIS

Vasiliki Stoumpou

External Supervisor: Theodoros Giannakopoulos
Principal Researcher NCSR Demokritos

NTUA Supervisor: Alexandros Potamianos
Associate Professor at the Electrical and Computer Engineering Department of NTUA

Athens, July 2021

.....
Βασιλική Στούμπου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα ποντίκια επικοινωνούν μεταξύ τους με υπερήχους (USVs (ultrasonic vocalizations)), οι οποία ποικίλλουν ανάλογα με το φύλο του ποντικιού και το στάδιο ανάπτυξης, τις περιβαλλοντικές και κοινωνικές συνθήκες και μπορούν να αξιοποιηθούν για τη μελέτη των νευρικών μηχανισμών που ενεργοποιούνται κατά την παραγωγή τους. Για να μελετηθεί η κοινωνική διάσταση των USVs, έχουν αναπτυχθεί εργαλεία για τον εντοπισμό και την ταξινόμησή τους σε διαφορετικές κατηγορίες. Πολλά από τα εργαλεία που έχουν ήδη αναπτυχθεί περιορίζονται σε offline προσεγγίσεις (περιορίζοντας τη δυνατότητα πειραμάτων πραγματικού χρόνου με ανατροφοδότηση), συχνά εξαρτώνται από συγκεκριμένες συνθήκες ηχογράφησης και χρησιμοποιούν επιβλεπόμενες προσεγγίσεις ταξινόμησης για την κατηγοριοποίηση των USVs, μειώνοντας την πιθανότητα ανακάλυψης νέων κλάσεων.

Για να αντιμετωπίσουμε αυτά τα ζητήματα, αναπτύξαμε ένα εργαλείο επεξεργασίας ηχογραφήσεων ποντικών που ονομάζεται AMVOC (Analysis of Mouse VOcal Communication), με αρχικό στόχο την ανίχνευση USVs (offline και online λειτουργία) με συνθήκες κατωφλίωσης στη φασματική ενέργεια. Συγκρίνοντας με δεδομένα που έχουν επισημειωθεί από ειδικούς, το AMVOC εμφάνισε υψηλή ακρίβεια στην ανίχνευση των USVs και ξεπέρασε τις πιο ευρέως διαδεδομένες μεθόδους σε θορυβώδεις συνθήκες. Η διαδικασία online δίνει αποτελέσματα τόσο ακριβή όσο η offline, ανοίγοντας τον δρόμο για πληθώρα νέων πειραμάτων.

Η πιο σημαντική συνεισφορά της δουλειάς μας είναι η εφαρμογή μιας μη επιβλεπόμενης μεθόδου βαθιάς μάθησης. Αυτή περιλαμβάνει τη συσταδοποίηση USVs χρησιμοποιώντας λανθάνουσες αναπαραστάσεις, που εξάγονται με χρήση ενός convolutional autoencoder. Η εντελώς μη επιβλεπόμενη προσέγγιση επιτρέπει εναλλακτικές ομαδοποιήσεις των USVs για την εξερεύνηση νέων κατηγοριών. Η διαδικασία αξιολόγησης έδειξε βελτίωση της ομαδοποίησης σε σύγκριση με την ομαδοποίηση με βάση χαρακτηριστικά που εξάγαμε με παραδοσιακές τεχνικές.

Μια άλλη καινοτομία της δουλειάς μας είναι η εφαρμογή μιας ημι-επιβλεπόμενης προσέγγισης για την ενίσχυση της ποιότητας της συσταδοποίησης. Η ημι-επίβλεψη γίνεται με τη μορφή περιορισμών σε ζευγάρια από USVs, οι οποίοι εκφράζουν κατά πόσο θέλουμε να ανήκουν στην ίδια ομάδα ή όχι. Η απόδοση της ομαδοποίησης αξιολογήθηκε και πάλι, δείχνοντας μια συνολική βελτίωση σε σχέση με την αρχική, μη επιβλεπόμενη ομαδοποίηση.

Η συσταδοποίηση που προκύπτει μπορεί να χρησιμοποιηθεί ως δεδομένα εκπαίδευσης ενός ταξινομητή, ο οποίος στη συνέχεια θα χρησιμοποιηθεί για την ταξινόμηση των USVs που ανιχνεύονται σε εφαρμογές πραγματικού χρόνου, παρέχοντας χρήσιμη ανατροφοδότηση σχετικά με τον τύπο κάθε USV.

Αυτά τα αποτελέσματα μπορούν να χρησιμοποιηθούν για να εξερευνήσουν το φωνητικό ρεπερτόριο των ποντικιών. Το AMVOC είναι ένα νέο εργαλείο που διευκολύνει τις φωνητικές αναλύσεις σε ένα ευρύτερο φάσμα πειραματικών συνθηκών και επιτρέπει στους χρήστες να αναπτύξουν νέα πειράματα με πρωτοφανείς διευκολύνσεις.

Λέξεις Κλειδιά: USVs ποντικιών, κοινωνική συμπεριφορά ποντικιών, μηχανική μάθηση, βαθιά μάθηση, εφαρμογές πραγματικού χρόνου, μη επιβλεπόμενη μάθηση, συσταδοποίηση, ομαδοποίηση, convolutional autoencoder, ημι-επιβλεπόμενη μάθηση

Abstract

Mice communicate using ultrasonic vocalizations (USVs), which vary depending on mouse gender and development stage, environmental and social conditions. Neural mechanisms underlying mouse ultrasonic vocalizations (USVs) are a useful model for the neurobiology of human speech and speech-related disorders, making their study an interesting, promising and popular field. To study social meaning and dimensions of mice vocalizations, tools have been developed to detect and classify USVs to different categories.

Many of the tools already developed are generally limited to offline approaches, are often dependent on specific recording conditions and employ supervised classification approaches to categorize vocalizations. These methods hinder real-time experimentation with produced vocalizations feedback and also limit the classification of vocalizations in pre-defined classes, reducing the likelihood of discovering new USVs types.

To address these issues, we developed a mice recordings processing tool called AMVOC (Analysis of Mouse VOcal Communication), whose functionalities include USVs detection in both offline and online modes, by using spectral energy thresholding. Compared to hand-annotated ground-truth data, AMVOC USVs detection functionality has high accuracy, and outperforms leading methods in noisy conditions, thus allowing for broader experimental uses. Real-time detection method gives results nearly as accurate as the offline procedure, also opening the door to numerous new experiments.

The most important contribution of our work is the implementation of an unsupervised deep learning method, which involves clustering USVs using latent features, extracted with the use of a convolutional autoencoder. The completely unsupervised approach we chose enables alternative groupings of vocalizations compared to predetermined classes, indicating the possibility of exploring new meaningful classes. Evaluation procedure involving human annotators showed an improvement of clustering USVs compared to clustering based on hand-crafted features.

Another novelty of our work is the application of a semi-supervised approach to enhance the clustering performance and propose an alternative clustering, more recording-focused. Human intervention comes in the form of pairwise constraints between vocalizations. The clustering performances were again evaluated, showing an overall refinement of the completely unsupervised clustering.

Resulting clustering can be used as ground-truth data to train a classifier, which will then be used to classify test data, i.e. vocalizations that are detected online, providing useful information about each vocalization's type in real-time experiments.

These results can be used to explore the vocal repertoire space of the analyzed vocalizations. AMVOC is a new tool that facilitates vocal analyses in a broader range of experimental conditions and allows users to develop previously inaccessible experimental designs in studies of mouse vocal behavior.

Key Words: mouse vocalization, ultrasonic vocalizations, social behavior, machine learning, deep learning, automated, real-time, unsupervised learning, clustering, convolutional autoencoder, semi-supervised learning

Ευχαριστίες

Θα ήθελα να ευχαριστήσω πρωτίστως τον κ. Θεόδωρο Γιαννακόπουλο, επιβλέποντα της εργασίας αυτής, του οποίου η βοήθεια, οι ιδέες και η καθοδήγηση σε κάθε βήμα ήταν απαραίτητες. Το ενδιαφέρον του ήταν συνεχές, η βοήθεια του πολύτιμη και η συνεργασία μας άψογη από την αρχή έως το τέλος.

Επίσης, τον υποφήφιο διδάκτορα César D. M. Vargas, για την παροχή των δεδομένων και για την ουσιαστική συμβολή του στις πληροφορίες βιολογικής φύσεως, καθώς και τη βοήθεια του σε όλα τα στάδια της εργασίας.

Ευχαριστώ πολύ τον υποψήφιο διδάκτορα Peter F. Schade για τις καίριες συμβουλές και παρεμβάσεις του.

Τέλος, θα ήθελα να ευχαριστήσω τους Rajvi Agravat, Elena Waidmann, Μαρίλη Στούμπου και Αριστοτέλη Συμπέθερο για τη βοήθειά τους στην αξιολόγηση των πειραματικών αποτελεσμάτων μας.

Contents

List of Figures

List of Tables

Chapter 0

Εκτεταμένη Ελληνική Περίληψη

0.1 Εισαγωγή

Η παρούσα διπλωματική εργασία πραγματεύεται την ανάλυση ηχογραφήσεων ποντικιών. Τα ποντίκια επικοινωνούν μεταξύ τους με υπερήχους (τα λεγόμενα USVs (ultrasonic vocalizations) (30-110 kHz)), η ανάλυση των οποίων μπορεί να μας δώσει πληροφορίες σχετικά με τα συναισθήματα, το περιβάλλον, το φύλο και το στάδιο ανάπτυξης των ποντικιών. Επίσης, μπορεί να αποτελέσει πηγή στοιχείων για τις υποκείμενες νευροβιολογικές διεργασίες της ομιλίας, οι οποίες αποτελούν ένα χρήσιμο μοντέλο για τη μελέτη της ανθρώπινης ομιλίας και διαταραχών ομιλίας.

Σε αυτό το πλαίσιο, κρίνεται χρήσιμη η ανάπτυξη εργαλείων για την ανίχνευση και την κατηγοριοποίηση των USVs, που θα μπορούν μετά να συνδεθούν με ενδεχόμενη κοινωνική συμπεριφορά των ποντικιών. Οι περιορισμοί της πλειοψηφίας των εργαλείων που έχουν αναπτυχθεί έως τώρα για αυτόν τον σκοπό είναι η δυσκολία γενίκευσης της ανίχνευσης των USVs σε ποικιλία συνθηκών (θόρυβος κλπ), το γεγονός πως λειτουργούν offline μειώνοντας τις δυνατότητες διεξαγωγής πειραμάτων σε πραγματικό χρόνο με ανατροφοδότηση ως προς τα παραγόμενα USVs και η χρήση προκαθορισμένων κλάσεων για την ταξινόμηση των τύπων των USVs, εμποδίζοντας την πιθανότητα κατηγοριοποίησης με βάση άλλα χαρακτηριστικά.

Στην εργασία αυτή αναπτύξαμε ένα ολοκληρωμένο εργαλείο επεξεργασίας και ανάλυσης USVs ποντικιών, που στόχος του είναι να ξεπεράσει τα προαναφερθέντα προβλήματα. Συγκεκριμένα, το εργαλείο που προτείνουμε έχει τις εξής δυνατότητες:

- Ανίχνευση των USVs με υψηλό ποσοστό ακρίβειας που συναγωνίζεται τις δημοφιλείς μεθόδους και μάλιστα τις ξεπερνά σε θορυβώδεις συνθήκες ηχογράφησης.
- Επέκταση της μεθόδου ανίχνευσης σε online εφαρμογές, παρέχοντας άμεση πληροφορία για τα ανιχνευθέντα USVs σε πραγματικό χρόνο.
- Μη επιβλεπόμενη συσταδοποίηση των USVs, βάσει χαρακτηριστικών που εξάγονται πάλι με μη επιβλεπόμενο τρόπο με τη χρήση ενός convolutional autoencoder. Η αξιολόγηση της συσταδοποίησης με τα χαρακτηριστικά που παρήχθησαν αυτόματα με βαθιά μάθηση συγκριτικά με παραδοσιακά χαρακτηριστικά που ορίστηκαν από εμάς και αφορούν τη μορφή των USVs έδειξε σημαντική υπεροχή της πρώτης.
- Δυνατότητα βελτίωσης της ποιότητας της συσταδοποίησης με χρήση ημιεπιβλεπόμενων τεχνικών που περιλαμβάνουν παρέμβαση του χρήστη στη διαδικασία της εκπαίδευσης. Και εδώ, η αξιολόγηση έδειξε βελτίωση της ποιότητας της συσταδοποίησης σε σχέση με την τελείως μη επιβλεπόμενη προσέγγιση.
- Χρήση των δεδομένων συσταδοποίησης (ζευγάρια: αναπαράσταση του USV - ομάδα στην οποία ανήκει) σαν δεδομένα εκπαίδευσης ενός ταξινομητή. Αυτός ο ταξινομητής

μπορεί να χρησιμοποιηθεί για την ταξινόμηση USVs που ανιχνεύονται σε εφαρμογές πραγματικού χρόνου.

0.2 Θεωρητικό υπόβαθρο

0.2.1 Επεξεργασία ακουστικών σημάτων

Τα σήματα είναι συναρτήσεις του χρόνου που χαρακτηρίζονται από το πλάτος τους και τη συχνότητά τους και μεταφέρουν ακουστική πληροφορία. Γενικά είναι αναλογικά, αλλά στους υπολογιστές αποθηκεύονται ψηφιακά σήματα που έχουν διακριτές τιμές στον χρόνο και διακριτές τιμές στο πλάτος, σαν μονοδιάστατοι πίνακες που στην πραγματικότητα αντιστοιχούν σε μια χρονική ακολουθία τιμών πλάτους.

Είναι σύνηθες τα σήματα να εξετάζονται και να υποβάλλονται σε επεξεργασία στο πεδίο της συχνότητας και όχι στο πεδίο του χρόνου, λόγω επιθυμητών ιδιοτήτων. Αυτό επιτυγχάνεται με τη χρήση του γνωστού μετασχηματισμού Fourier, που μετασχηματίζει το σήμα από μία ακολουθία τιμών στο πεδίο του χρόνου, σε μία ακολουθία συντελεστών στο πεδίο της συχνότητας (φάσμα).

Επειδή οι ιδιότητες και τα χαρακτηριστικά των σημάτων μεταβάλλονται με την πάροδο του χρόνου, είναι συχνά χρήσιμο αντί να μετασχηματίζεται όλο το σήμα απευθείας από τον χρόνο στη συχνότητα, να το χωρίζουμε σε χρονικά πλαίσια, και να εφαρμόζουμε τον μετασχηματισμό Fourier σε κάθε πλαίσιο μεμονωμένα. Αυτή η μέθοδος ονομάζεται STFT (Short Term Fourier Transform). Οι ακολουθίες που προκύπτουν στο πεδίο της συχνότητας μπορούν να τοποθετηθούν κάθετα η μία δίπλα στην άλλη για κάθε χρονικό πλαίσιο, έτσι ώστε να σχηματιστεί ένας δισδιάστατος πίνακας, όπου ο οριζόντιος άξονας αντιστοιχεί στον χρόνο και ο κάθετος στη συχνότητα. Οι τιμές του εκφράζουν την ενέργεια που έχει το σήμα σε κάθε χρονικό πλαίσιο και σε κάθε τιμή συχνότητας. Αυτό είναι το λεγόμενο φασματογράφημα του σήματος, που περιέχει χρονική και συχνοτική πληροφορία και μας δίνει τη δυνατότητα να έχουμε την εποπτεία των σημάτων σαν εικόνες στον δισδιάστατο χώρο.

0.2.2 Μηχανική μάθηση - CNNs, Autoencoders, Clustering, SDEC

Η μηχανική μάθηση είναι ένας ραγδαία αναπτυσσόμενος τομέας που εστιάζει στην ταξινόμηση προτύπων (επιβλεπόμενη μάθηση) σε συγκεκριμένες κλάσεις ή στην ομαδοποίηση προτύπων όταν οι κλάσεις δεν είναι εκ των προτέρων γνωστές (μη επιβλεπόμενη μάθηση). Επίσης, υπάρχει και η δυνατότητα της ημιεπιβλεπόμενης μάθησης, που περιλαμβάνει μερική γνώση για τα δεδομένα, η οποία αξιοποιείται ανάλογα με την εφαρμογή.

Η βαθιά μάθηση είναι ένας κλάδος της μηχανικής μάθησης που στοχεύει όχι μόνο στην ταξινόμηση προτύπων, αλλά και στην εξεύρεση των χαρακτηριστικών τους που είναι χρήσιμα για αυτόν τον σκοπό. Όταν αναφερόμαστε στη βαθιά μάθηση, κυρίως αναφερόμαστε στα νευρωνικά δίκτυα (στρώματα από επεξεργαστικές μονάδες, τους νευρώνες, που συνδέονται μεταξύ τους).

Ένας ευρέως χρησιμοποιούμενος τύπος νευρωνικών δίκτυων είναι τα συνελικτικά νευρωνικά δίκτυα (CNNs), τα οποία δέχονται στην είσοδό τους εικόνες και μαθαίνουν τα σημαντικά χαρακτηριστικά τους, δίνοντας στην έξοδο τους αναπαραστάσεις αυτών των εικόνων. Τα CNNs αποτελούνται από δύο είδη στρωμάτων. Το πρώτο είναι τα συνελικτικά στρώματα που εφαρμόζουν την πράξη της δισδιάστατης συνέλιξης χρησιμοποιώντας φίλτρα στην αρχική εικόνα, των οποίων τα βάρη είναι προς εκμάθηση, με στόχο την εκμάθηση τοπικών χαρακτηριστικών της εικόνας. Το δεύτερο είναι τα στρώματα "pooling" που λειτουργούν σαν υποδειγματοληψία της εξόδου του συνελικτού στρώματος. Με τη διαδικασία αυτή, μειώνουμε τον υπολογιστικό φόρτο των επόμενων στρωμάτων, καθώς θα πρέπει να διαχειριστούν εικόνες μικρότερου μεγέθους, ενώ αποφεύγεται και το overfitting, λόγω της μείωσης των απαιτού-

μενων παραμέτρων προς εκπαίδευση. Επίσης, καθώς μειώνονται οι διαστάσεις των εικόνων, μικροί μετασχηματισμοί, αλλοιώσεις ή παραμορφώσεις της εικόνας δεν παίζουν ρόλο. Έτσι, παίρνουμε μία αναπαράσταση της εικόνας πρακτικά ανεξάρτητη της κλίμακας.

Τα CNNs μπορούν να χρησιμοποιηθούν ως μέρος των autoencoders για την εξαγωγή χρήσιμων αναπαραστάσεων. Οι autoencoders είναι μία κατηγορία βαθιών νευρωνικών δικτύων που χρησιμοποιούνται στη μη επιβλεπόμενη μάθηση για να μάθουν λανθάνουσες αναπαραστάσεις (χαρακτηριστικά) των προτύπων εισόδου, όταν δεν είναι γνωστές π.χ οι κλάσεις στις οποίες ανήκουν αυτά τα πρότυπα. Συγκεκριμένα, αποτελούνται από έναν κωδικοποιητή (encoder) ο οποίος δίνει στην έξοδο μια ενδιάμεση αναπαράσταση (code) και έναν αποκωδικοποιητή (decoder), του οποίου ο στόχος είναι να χρησιμοποιήσει αυτή την αναπαράσταση για την ανακατασκευή της αρχικής εισόδου. Με τη σύγκριση της εξόδου του αποκωδικοποιητή με την αρχική είσοδο επιτυγχάνεται η εκπαίδευση του autoencoder.

Όταν όντως τα CNNs συνδυάζονται με τους autoencoders (convolutional autoencoders), τότε και ο encoder και ο decoder είναι πρακτικά CNNs. Ο encoder μαθαίνει χαρακτηριστικά από τις εικόνες εισόδου και δίνει στην έξοδό του τις ενδιάμεσες αναπαραστάσεις. Ο decoder χρησιμοποιεί την ενδιάμεση αναπαράσταση για την ανακατασκευή της εικόνας εισόδου.

Τα χαρακτηριστικά που παράγονται στην έξοδο του encoder, μπορούν να χρησιμοποιηθούν για συσταδοποίηση (clustering) των προτύπων. Υπάρχουν διάφοροι αλγόριθμοι συσταδοποίησης που ακολουθούν διαφορετικές προσεγγίσεις (K-Means, GMMs, Agglomerative κλπ), οι οποίοι δημιουργούν μια ομαδοποίηση των προτύπων με βάση τα χαρακτηριστικά τους.

Αυτές οι αναπαραστάσεις, καθώς και η συνεπαγόμενη ομαδοποίηση μπορεί να βελτιωθούν περαιτέρω με τη χρήση ημιεπιβλεπόμενης μάθησης. Μια υλοποίηση αυτής της ιδέας είναι το SDEC (Semi-supervised Deep Embedded Clustering). Αυτός ο αλγόριθμος επανεκπαιδεύει έναν ήδη εκπαιδευμένο encoder, όπου τώρα αντί για την ανακατασκευή της εισόδου από τον decoder, η εκπαίδευση γίνεται με στόχο 1) την ομογενοποίηση των συστάδων (clusters) που σχηματίζονται από τις αναπαραστάσεις των τρεχόντων εξόδων του encoder και 2) την ενσωμάτωση κάποιας μορφής πληροφορίας που δίνουμε εμείς. Στην προκειμένη περίπτωση, αυτή έχει τη μορφή περιορισμών που τίθενται μεταξύ προτύπων: αν πρέπει να ανήκουν στην ίδια ομάδα ή όχι. Αυτοί οι δύο παράγοντες κατευθύνουν την επανεκπαίδευση του encoder με στόχο τη βελτίωση των αναπαραστάσεων και της συσταδοποίησης.

0.2.3 Μετρικές αξιολόγησης

Σε ένα πρόβλημα ταξινόμησης, η αξιολόγηση του μοντέλου γίνεται με τη χρήση του NxN πίνακα σύγχησης (confusion matrix, N ο αριθμός των κλάσεων), ο οποίος αποτελείται στη συνηθισμένη περίπτωση των 2 διαστάσεων από 4 στοιχεία: TP (true positive) είναι ο αριθμός των προτύπων που προβλέφθηκε ότι σωστά ανήκουν στην κλάση 1, TN (true negative) είναι ο αριθμός των προτύπων που σωστά προβλέφθηκε ότι ανήκουν στην κλάση 0, FP (false positive) είναι ο αριθμός των προτύπων που προβλέφθηκε ότι ανήκουν στην κλάση 1 ενώ στην πραγματικότητα ανήκουν στην κλάση 0 και FN (false negative) είναι ο αριθμός των προτύπων που προβλέφθηκε ότι ανήκουν στην κλάση 0 ενώ στην πραγματικότητα ανήκουν στην κλάση 1.

Οι δύο βασικές μετρικές που χρησιμοποιούνται είναι η ανάκληση (recall) (ποσοστό των προτύπων που ανήκουν στην κλάση 1 και προβλέφθηκε ότι όντως ανήκουν σε αυτή, $R = \frac{TP}{TP+FN}$) και η ακρίβεια (precision) (ποσοστό των προτύπων που προβλέφθηκε ότι ανήκουν στην κλάση 1 και όντως ανήκουν σε αυτή, $P = \frac{TP}{TP+FP}$). Συχνά, προκειμένου να έχουμε μια αντικειμενική εικόνα και να λαμβάνουμε υπόψη και τις δύο μετρικές χρησιμοποιούμε τον αρμονικό μέσο τους (F1 score, $F1 = \frac{2PR}{P+R}$). Σε προβλήματα με πολλαπλές κλάσεις και μη ισορροπημένα δεδομένα, υπολογίζουμε το F1 score κάθε κλάσης και υπολογίζουμε τον μέσο όρο τους (macro F1 score).

Αυτή την προσέγγιση μπορούμε να ακολουθήσουμε και για να συγκρίνουμε ανιχνευθέντα

γεγονότα (π.χ. USVs) από δύο μεθόδους, χρησιμοποιώντας το χρονικό διάστημα που έχει ανατεθεί σε κάθε γεγονός. Υπάρχουν δύο διαφορετικοί τρόποι να αξιοποιήσουμε τον πίνακα σύγχυσης: η χρονική αξιολόγηση (temporal evaluation), στην οποία χωρίζουμε τον χρόνο σε μικρές χρονικές μονάδες, στις οποίες έχουμε (κλάση 1) ή όχι (κλάση 0) κάποιο γεγονός (πρόβλημα ταξινόμησης κάθε χρονικής μονάδας σε 2 κλάσεις) και η αξιολόγηση γεγονότων (event evaluation), όπου ως TP ορίζουμε τα ανχινευθέντα γεγονότα των οποίων το χρονικό διάστημα είχε επικάλυψη με ένα πραγματικό γεγονός, ως FP τα ανιχνευθέντα γεγονότα που δεν είχαν επικάλυψη με πραγματικά και ως FN είναι τα γεγονότα που απέτυχαν αν ανιχνευθούν. Με αυτή την προασρμογή μπορούμε να υπολογίσουμε πλέον οποιαδήποτε μετρική.

0.3 Δεδομένα

Στη διάρκεια των πειραματικών διαδικασιών αξιολόγησης, χρησιμοποιήσαμε τρία σύνολα δεδομένων (D1, D2 και D3).

Το σύνολο δεδομένων D1 δημιουργήθηκε για την αξιολόγηση των μεθόδων ανίχνευσης των USVs. Πρόκειται για ένα σύνολο από 9 ακουστικά κομμάτια διάρκειας 5-10 δευτερολέπτων το καθένα, που περιέχει 245 USVs συνολικά. Αυτά τα USVs επισημειώθηκαν από ειδικό, ο οποίος όρισε την αρχή και το τέλος του κάθε USV, με χρονική ανάλυση 1 ms.

Το σύνολο δεδομένων D2 αποτελείται από 26 διαφορετικά αρχεία ήχου και χρησιμοποιήθηκε για την εκπαίδευση του convolutional autoencoder μας.

Το σύνολο δεδομένων D3 χρησιμοποιήθηκε για την αξιολόγηση της διαδικασίας συσταδοποίησης. Συγκεκριμένα, χρησιμοποιήσαμε ένα σύνολο από 72 αρχεία ήχου, διαλέξαμε 20 δευτερόλεπτα από το καθένα που φροντίσαμε να περιέχουν τουλάχιστον 2.5 USVs ανά δευτερόλεπτο και στη συνέχεια συνενώσαμε διαδοχικά αυτά τα κομμάτια, σχηματίζοντας συνολικά 4 νέα αρχεία ήχου.

0.4 Ανίχνευση USVs

Η πρώτη λειτουργία που υλοποιεί το AMVOC είναι η ανίχνευση των USVs των ποντικιών. Για τον σκοπό αυτό, χρησιμοποιούμε το φασματογράφημα που υπολογίζουμε από το σήμα ήχου, το οποίο γενικά επεξεργαζόμαστε μόνο στο φάσμα ενδιαφέροντος (30-110 kHz). Η χρονική ανάλυση είναι 2 ms και η συχνοτική 0.5 kHz. Έχουμε υλοποιήσει τη μέθοδό μας και για offline εφαρμογές με τη χρήση ήδη ηχογραφημένων σημάτων, καθώς και για εφαρμογές πραγματικού χρόνου, όπου το σήμα καταγράφεται και η ανίχνευση γίνεται ταυτόχρονα.

0.4.1 Offline λειτουργία

Αφού υπολογιστεί το φασματογράφημα, η ανίχνευση των USVs γίνεται με την εφαρμογή δύο κριτηρίων.

- Το πρώτο κριτήριο αφορά κατωφλίωση της φασματικής ενέργειας ανά χρονικό πλαίσιο. Πιο συγκεκριμένα, για κάθε χρονικό πλαίσιο αθροίζουμε όλες τις τιμές του φασματογραφήματος E που αντιστοιχούν στο πεδίο των συχνοτήτων (φάσμα):

$$S_i = \sum_{j=30kHz}^{110kHz} E_{ij} \tag{0.1}$$

  όπου ο δείκτης j αυξάνεται κατά 0.5 kHz. Έτσι, για κάθε χρονικό πλαίσιο i έχουμε την τιμή της φασματικής ενέργειας S_i. Στη συνέχεια, υπολογίζουμε μια δυναμική ακολουθία

κατωφλίων για κάθε χρονικό πλαίσιο i ως εξής:

$$T_i = \frac{1}{2}\frac{\sum_{j=0}^{N-1} S_j}{N} + \frac{1}{2}\frac{\sum_{j=0}^{K-1} S_{i-j}}{K} \tag{0.2}$$

όπου N είναι ο αριθμός των χρονικών πλαισίων από τα οποία αποτελείται το φασματογράφημα, και K το μέγεθος ενός φίλτρου κινούμενου μέσου που εφαρμόζουμε στο σήμα ($K = 2$ στην περίπτωσή μας). Συνεπώς ο πρώτος όρος αντιστοιχεί στη μέση φασματική ενέργεια όλου του σήματος και ο δεύτερος στην τιμή της φασματικής ενέργειας το χρονικό πλαίσιο i, αφού η ενέργεια συνελιχθεί με ένα φίλτρο κινούμενου μέσου για λόγους εξομάλυνσης (επομένως η νέα τιμή στο πλαίσιο i καθορίζεται και από τις K προηγούμενες).

- Το δεύτερο κριτήριο αφορά την κατανομή της ενέργειας σε κάθε χρονικό πλαίσιο στο φασματογράφημα. Αν έχουμε υψηλές τιμές του φασματογραφήματος γύρω από μια συχνότητα, πιθανότατα αυτές οφείλονται στην παρουσία θορύβου. Γι' αυτό, υπολογίζουμε ένα νέο δυναμικό κατώφλι, που θα εφαρμόσουμε στη μέγιστη τιμή της ενέργειας ανά χρονικό πλαίσιο $P_i = \max_{j=30,\ldots,110kHz} E_{ij}$. Αν ορίσουμε ως p_i τη συχνότητα της μέγιστης ενέργειας στο χρονικό πλαίσιο i, η δυναμική ακολουθία κατωφλίων ορίζεται από:

$$M_i = \frac{1}{N_f} \sum_{j=\max(p_i-30kHz,30kHz)}^{\min(p_i+30kHz,110kHz)} E_{ij} \tag{0.3}$$

που εκφράζει τη μέση ενέργεια στις συχνότητες γύρω από τη μέγιστη ($N_f = 2\cdot(\min(p_i+30kHz, 110kHz) - \max(p_i - 30kHz, 30kHz))$ εφόσον ο μετρητής j αυξάνεται κατά 0.5 kHz).

Οι συνθήκες κατωφλίωσης εφαρμόζονται ως εξής:

$$V_i = \begin{cases} 1 & \text{αν } (S_i > t \cdot T_i) \text{ ΚΑΙ } (P_i > f \cdot M_i), \text{ όπου } t = 0.5 \text{ και } f = 3.5 \\ 0 & \text{αλλιώς} \end{cases} \tag{0.4}$$

Έτσι, προκύπτει η ακολουθία V, που αποτελείται από 0 και 1 τα οποία εκφράζουν αν το τρέχον χρονικό πλαίσιο αντιστοιχεί σε USV ή όχι. Αυτή η ακολουθία στη συνέχεια φιλτράρεται με ένα φίλτρο κινούμενου μέσου (διάρκειας 20 ms) για να ληφθεί υπόψη η γειτονιά ενός πιθανού USV. Τέλος, ενώνουμε διαδοχικά USVs που απέχουν κάτω από 11 ms και απορρίπτουμε όσα έχουν διάρκεια κάτω από 5 ms, καθώς συνήθως πρόκειται για θόρυβο.

0.4.2 Online λειτουργία

Στη λειτουργία πραγματικού χρόνου, έχουμε τη δυνατότητα να παρέχουμε feedback στον χρήστη για τα ανιχνευθέντα USVs τη στιγμή της ηχογράφησης. Η διαδικασία που ακολουθούμε είναι ίδια με την offline λειτουργία, η μοναδική διαφορά είναι ο ορισμός της πρώτης ακολουθίας κατωφλίων T_i, δεδομένου ότι τώρα δεν μπορούμε να λάβουμε υπόψη όλο το σήμα γιατί δεν είναι ακόμα ηχογραφημένο. Συγκεκριμένα, επεξεργαζόμαστε το σήμα ανά κάθε ηχογραφημένο τμήμα διάρκειας 750 ms, το οποίο θέλαμε να είναι μικρό για να μπορούμε να παρέχουμε γρήγορο feedback, ωστόσο να είναι αρκετά μεγάλο για να έχουν αξιοπιστία οι τιμές που χρησιμοποιούνται για τον ορισμό των δυναμικών κατωφλίων.

Η βασική τους διαφορά είναι ότι τώρα η πρώτη ακολουθία κατωφλίων έχει την ίδια τιμή για όλα τα χρονικά πλαίσια ενός παραθύρου επεξεργασίας 750 ms και ορίζεται για κάθε παράθυρο k ως:

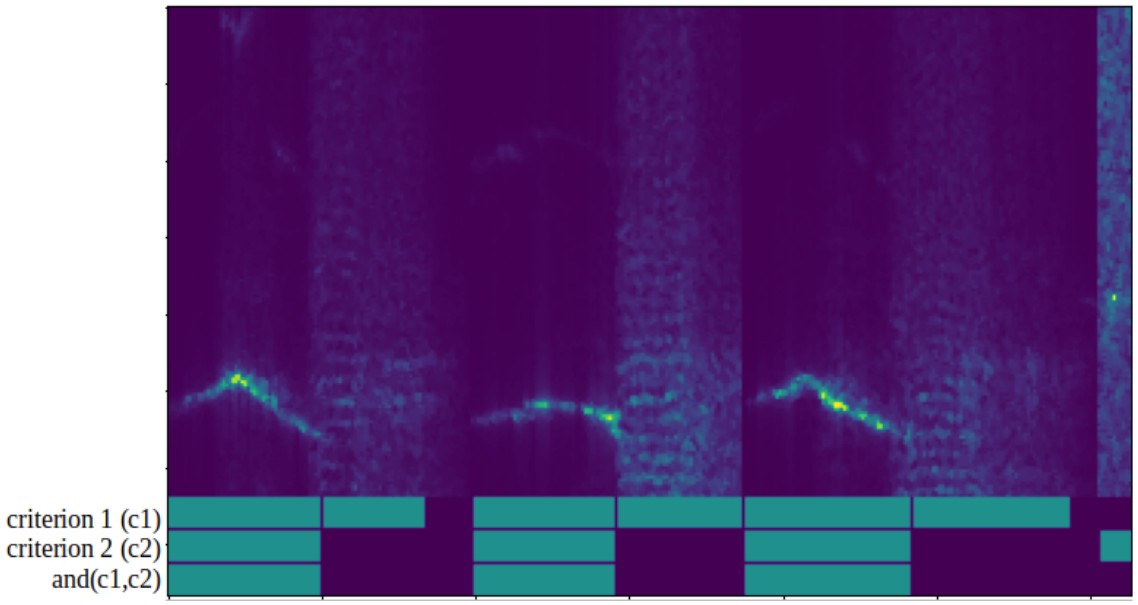


Figure 0.1: Παραδείγματα χρήσης των δύο κριτηρίων κατωφλίωσης. Οι πράσινες μπάρες των δύο πρώτων γραμμών δείχνουν τα USVs που ανιχνεύονται από κάθε κριτήριο, ενώ οι μπάρες της τρίτης γραμμής την τομή τους.

$$T_k = 0.3\frac{\sum_{j=1}^{k} B_j}{k} + 0.7 \cdot B_k \tag{0.5}$$

όπου B_k είναι η μέση φασματική ενέργεια του παραθύρου (block) k. Άρα το κατώφλι τώρα υπολογίζεται ως το σταθμισμένο άθροισμα των μέσων τιμών ενέργειας όλων των προηγούμενων παραθύρων και της μέσης ενέργειας του τρέχοντος παραθύρου. Τα βάρη (0.3, 0.7) επιλέχθηκαν μετά από πειραματισμό.

Με τη χρήση των παραθύρων των 750 ms, ουσιαστικά διακόπτουμε το σήμα σε κάποιο χρονικό σημείο το οποίο ενδέχεται να έχουμε ένα USV. Αυτά τα USVs δεν ανιχνεύονται με αυτή την προσέγγιση. Για τον λόγο αυτό, εκτός από τα 750 ms του τρέχοντος παραθύρου επεξεργαζόμαστε κάθε φορά ξανά και τα τελευταία 100 ms του προηγούμενου παραθύρου, ώστε να εξετάσουμε ενδεχόμενο USV που χάθηκε λόγω του τέλους του προηγούμενου παραθύρου.

Αξίζει να αναφερθεί ότι τόσο η offline, όσο και η online λειτουργία επιτρέπουν την επέμβαση του χρήστη με την επιλογή των συντελεστών κατωφλίων t και f. Οι τιμές που επιλέξαμε εμείς ορίστηκαν με βάση δοκιμές με το σύνολο δεδομένων D1. Αύξηση της τιμής αυτών των δύο παραμέτρων έχει ως αποτέλεσμα πιο αυστηρή ανίχνευση (αύξηση των FN λόγω του ότι ίσως αγνοούνται κάποια USVs με πιο χαμηλή ενέργεια ή σε περίπτωση θορυβωδών συνθηκών) ενώ μείωσή τους σημαίνει πιο "χαλαρή" ανίχνευση (αύξηση των FP λόγω της μεγαλύτερης πιθανότητας ανίχνευσης θορύβου).

0.4.3 Πειραματική αξιολόγηση της ανίχνευσης των USVs

Η αξιολόγηση της μεθοδολογίας ανίχνευσης των USVs έγινε συγκρίνοντας την απόδοση τόσο της offline όσο και της online προσέγγισης με άλλα εργαλεία και συγκεκριμένα τα MSA (Mouse Song Analyzer-2 εκδόσεις) (Arriaga et al. 2012; Chabout et al. 2015)), MUPET (Van Segbroeck et al. 2017), VocalMat (Fonseca et al. 2021), και DeepSqueak (Coffey et al. 2019), τα οποία τρέξαμε με τις προκαθορισμένες (default) παραμέτρους τους. Για τη σύγκριση χρησιμοποιήσαμε το σύνολο δεδομένων D1 που αποτελείται από 9 τμήματα διάρκειας

5-10 s έκαστο, που περιέχει 245 επισημειωμένα USVs συνολικά. Χωρίσαμε αυτό το σύνολο δεδομένων στα ηχογραφημένα τμήματα με και χωρίς θόρυβο.

Για την αξιολόγηση, χρησιμοποιήσαμε τη χρονική αξιολόγηση (temoral evaluation) και την αξιολόγηση γεγονότων (event evaluation). Τα αποτελέσματα φαίνονται στον Πίνακα 0.1.

		AMVOC						
F1 score		offline	online	MSA1	MSA2	MUPET	VocalMat	DeepSqueak
Temporal	Normal	84	**85**	46	88	85	91	83
	Noisy	67	68	23	71	53	57	**76**
	Average	75.5	76.5	34.5	**79.5**	69	74	**79.5**
Event	Normal	**97**	**97**	66	94	93	90	93
	Noisy	**84**	83	33	72	57	58	81
	Average	**90.5**	90	49.5	83	75	74	87

Table 0.1: F1 scores της μεθόδου του AMVOC και άλλων μεθόδων.

Παρατηρούμε ότι το AMVOC έχει καλύτερο event F1 score και στα θορυβώδη και στα καθαρά σήματα, ενώ οι μέθοδοι MSA2 και DeepSqueak είχαν λίγο καλύτερη απόδοση στη μετρική temporal F1, κυρίως λόγω των πιο εύστοχων ανιχνεύσεων στα θορυβώδη σημεία.

Θεωρήσαμε ενδιαφέρουσα και τη μελέτη της χρονικής απόδοσης των μεθόδων και για αυτό υπολογίσαμε το Realtime processing ratio, που ουσιαστικά ισούται με τον λόγο της πραγματικής διάρκειας ενός σήματος διά τον χρόνο που χρειάζόμαστε για την επεξεργασία του σήματος και την ανίχνευση των USVs. Προφανώς, όσο μεγαλύτερη είναι αυτή η μετρική, τόσο το καλύτερο. Για παράδειγμα, αν $r = 30$, σημαίνει ότι η μέθοδος είναι 30 φορές πιο γρήγορη από τον πραγματικό χρόνο, δηλαδή ότι χρειάζεται 1 λεπτό για να επεξεργαστούμε 30 λεπτά ακουστικής πληροφορίας. Τα αποτελέσματα φαίνονται στον πίνακα 0.2.

Methods	Real-time Processing Ratio
MUPET	**32.4**
MSA2	29.9
MSA1	28.1
AMVOC	21.2
DeepSqueak	8.2
VocalMat	4.3

Table 0.2: Real-time Processing Ratio. Τα πειράματα για αυτούς τους υπολογισμούς έγιναν για 5 διαφορετικές αρχεία ήχου, 3 φορές για το καθένα, και υπολογίστηκε ο μέσος όρος για κάθε μέθοδο.

Το AMVOC είχε ενδιάμεση επίδοση όσον αφορά αυτή τη μετρική. Την πιο γρήγορη ανίχνευση πετυχαίνουν το MUPET και το MSA.

Αξίζει επίσης να σημειωθεί, ότι η επίδοση της online λειτουργίας του AMVOC συναγωνίζεται τις αποδόσεις των πιο δημοφιλών εργαλείων και δεν υστερεί σε σχέση με την offline λειτουργία.

0.5 Βαθιά εξαγωγή χαρακτηριστικών με Convolutional Autoencoder

Αφού ανιχνευθούν τα USVs, θέλουμε να τα κατατάξουμε σε κλάσεις ή να τα ομαδοποιήσουμε με βάση κάποια χαρακτηριστικά τους. Η προσέγγισή μας είναι μη επιβλεπόμενη, δηλαδή θέλουμε να εξάγουμε χαρακτηριστικά από τη μορφή των φασματογραφημάτων των USVs και στη συνέχεια να ομαδοποιήσουμε τα USVs. Για την οπτικοποίηση της ομαδοποίησης και τη δυνατότητα αξιολόγησής της, έχουμε υλοποιήσει μια διεπαφή, στην οποία ο χρήστης μπορεί να επιλέξει και διαφορετικές ρυθμίσεις για τη συσταδοποίηση (αλγόριθμο, αριθμό συστάδων).

Για τη βαθιά εξαγωγή χαρακτηριστικών χρησιμοποιήσαμε έναν conolutional autoencoder. Αυτός δέχεται στην είσοδο τα φασματογραφήματα των USVs, και η έξοδος του encoder μας δίνει τα διανύσματα χαρακτηριστικών.

0.5.1 Αρχιτεκτονική και Εκπαίδευση του Convolutional Autoencoder

Για την εκπαίδευση του μοντέλου μας χρησιμοποιήσαμε το σύνολο δεδομένων D2, που αποτελείται από 22,409 USVs. Οι διαστάσεις των εικόνων πρέπει να είναι ίδιες, το οποίο ισχύει για τον άξονα συχνοτήτων (30-110 kHz), αλλά όχι για τον άξονα του χρόνου. Πρέπει να επιλέξουμε σταθερό μέγεθος για τον άξονα του χρόνου και να εξισορροπήσουμε την πιθανή απώλεια πληροφορίας αν ορίσουμε μικρό αριθμό χρονικών πλαισίων και "κόψουμε" τις εικόνες με τη μείωση της σημασίας των χαρακτηριστικών των μικρότερων σε χρονική διάρκεια USVs αν επεκτείνουμε με μηδενικά τις εικόνες. Επιλέξαμε να ορίσουμε σταθερό μέγεθος τα 64 χρονικά πλαίσια που αντιστοιχούν σε 128 ms, λαμβάνοντας υπόψη τη διάρκεια της πλειοψηφίας των USVs. Άρα, λαμβάνοντας υπόψη τη χρονική και συχνοτική ανάλυση (resolution), οι εικόνες έχουν διαστάσεις 64x160 και εισέρχονται στο μοντέλο σε batches.

Όσον αφορά την αρχιτεκτονική, φαίνεται στη γραφική παράσταση 0.2. Και ο encoder και ο decoder αποτελούνται από 3 συνελικτικά στρώματα με 64, 32 και 8 φίλτρα διάστασης 3x3 αντίστοιχα (με την αντίστροφη σειρά για τον decoder), τα οποία ακολουθούνται από max pooling στρώματα, που μειώνουν τη διάσταση της εξόδου των συνελικτικών δικτύων στο μισό. Χρησιμοποιούμε ως συνάρτηση ενεργοποίησης τη ReLU, εκτός από το τελευταίο στρώμα που χρησιμοποιείται η σιγμοειδής, με στόχο να δώσει τιμή μεταξύ 0 και 1 σε κάθε θέση της εξόδου. Ως συνάρτηση σφάλματος χρησιμοποείται η Binary Cross Entropy, που έχει στόχο να μειώσει την απόσταση μεταξύ της ανακατασκευασμένης εικόνας και της εικόνας εισόδου.

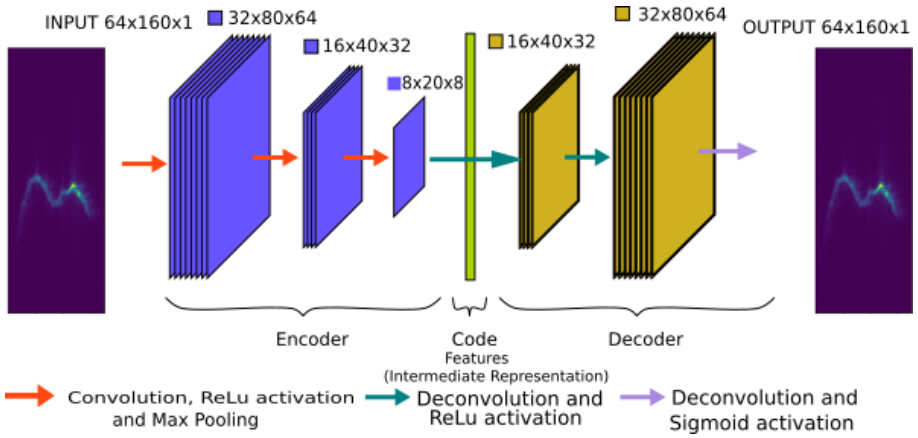


Figure 0.2: AMVOC convolutional autoencoder: Αρχιτεκτονική του δικτύου που χρησιμοποιήσαμε.

Όλες οι παράμετροι (αριθμός στρωμάτων, διαστάσεις φίλτρων) επιλέχθηκαν μετά από πειραματισμό. Το μοντέλο μας εκπαιδεύτηκε για 2 εποχές, καθώς παρατηρήσαμε ότι το

σφάλμα δεν εμφανίζει σημαντική μείωση μετά τις 2 εποχές, ενώ θέλαμε να αποφύγουμε και ενδεχόμενο overfitting στα δεδομένα εκπαίδευσης.

0.5.2 Βαθιά Εξαγωγή χαρακτηριστικών

Αφού το μοντέλο εκπαιδευτεί, ο encoder μπορεί να χρησιμοποιηθεί για την εξαγωγή χαρακτηριστικών. Επομένως, για ένα οποιοδήποτε ηχητικό σήμα, αφού υπολογιστεί το φασματογράφημα και ανιχνευτούν τα USVs, τα φασματογραφήματά τους εισέρχονται ως εικόνες στον encoder και η έξοδός του δίνει ένα διάνυσμα χαρακτηριστικών για κάθε εικόνα εισόδου.

Λόγω της μεγάλης διάστασης αυτού του διανύσματος, περνάμε τις αναπαραστάσεις από μία επεξεργασία πριν τις χρησιμοποιήσουμε για συσταδοποίηση. Αυτή περιλαμβάνει:

1. Την αφαίρεση χαρακτηριστικών που έχουν μικρή διακύμανση στο σύνολο των αναπαραστάσεων των USVs που εξετάζουμε.

2. Τον μετασχηματισμό των εναπομεινάντων χαρακτηριστικών αφαιρώντας τη μέση τιμή κάθε χαρακτηριστικού και διαιρώντας με τη διασπορά.

3. Τη μείωση των διαστάσεων με τη μέθοδο PCA, η οποία μετασχηματίζει τα διανύσματα χαρακτηριστικών σε έναν χώρο με μικρότερες διαστάσεις, διατηρώντας όσο το δυνατόν περισσότερο τη διακύμανση των αρχικών χαρακτηριστικών.

0.5.3 Παραδοσιακή εξαγωγή χαρακτηριστικών

Προκειμένου να αξιολογήσουμε τις αναπραστάσεις που πήραμε από τη βαθιά εξαγωγή χαρακτηριστικών, θέλουμε να συγκρίνουμε την επίδοσή τους με χαρακτηριστικά που επιλέγονται από εμάς και αφορούν βασικά τη μορφή των φωνητικών "συλλαβών" όπως αποτυπώνονται στο φασματογράφημα. Η πληροφορία λαμβάνεται από μία δισδιάστατη καμπύλη όπου ο άξονας x αντιστοιχεί στον χρόνο (στα χρονικά πλαίσια) και ο y στη συχνότητα, με τις τιμές της καμπύλης να δίνονται από τον τύπο $y_i = \arg\max_j E_{ij}$, δηλαδή με τις τιμές συχνοτήτων όπου η ενέργεια παίρνει τη μέγιστη τιμή σε κάθε χρονικό πλαίσιο του φασματογραφήματος. Συγκεκριμένα, αυτά αφορούν τη 1) διάρκεια κάθε USV, 2) το χρονικό σημείο της ελάχιστης τιμής της καμπύλης, 3) το χρονικό σημείο της μέγιστης τιμής της καμπύλης και 4) τη διαφορά μεταξύ μεγίστου και ελαχίστου της καμπύλης. Κάθε χαρακτηριστικό μετασχηματίζεται αφαιρώντας του τη μέση τιμή που του αντιστοιχεί και διαιρώντας με τη διασπορά.

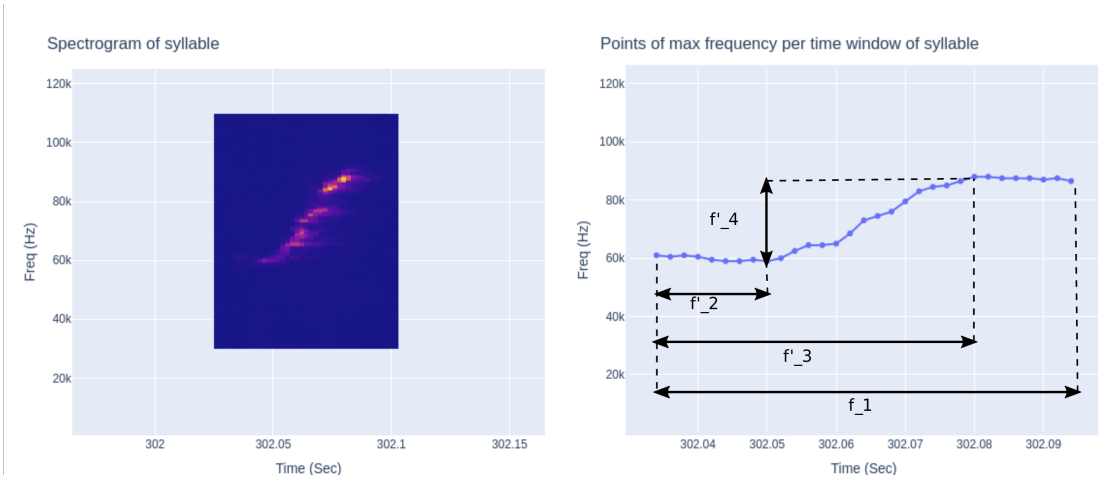


Figure 0.3: Ένα USV (αριστερά) και η καμπύλη με τις συχνότητες μέγιστης ενέργειας (δεξιά), που χρησιμοποιείται για την εξαγωγή των παραδοσιακών χαρακτηριστικών.

0.5.4 Συσταδοποίηση

Για τη συσταδοποίηση μπορούμε να χρησιμοποιήσουμε διάφορους αλγορίθμους που ακολουθούν διαφορετικές προσεγγίσεις. Αυτοί που επιλέξαμε είναι οι Agglomerative, Gaussian Mixture Models, K-Means, Mini-Batch K-Means και Birch. Τα αποτελέσματα της συσταδοποίησης φαίνονται στη γραφική διεπαφή που έχουμε αναπτύξει.

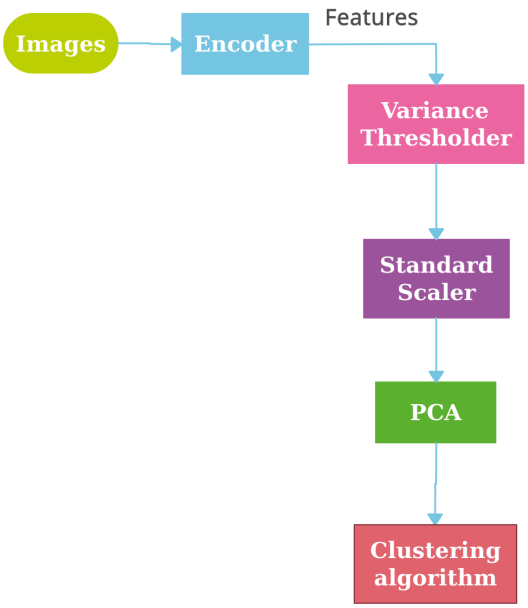


Figure 0.4: Σύνοψη της διαδικασίας βαθιάς εξαγωγής χαρακτηριστικών και συσταδοποίησης.

0.5.5 Πειραματική αξιολόγηση της συσταδοποίησης με το AMVOC

Η αξιολόγηση έγινε συγκρίνοντας συγκεκριμένα setups συσταδοποίησης όπως προέκυψαν με τις δύο κατηγορίες χαρακτηριστικών (βαθιά και παραδοσιακά). Οι διατάξεις που χρησιμοποιήσαμε είναι οι 1) Agglomerative clustering με 6 clusters, Gaussian Mixture με 6 clusters και K-Means με 6 clusters. Τέσσερις χρήστες βαθμολόγησαν αυτά τα setups στα 4 αρχεία του συνόλου δεδομένων D3.

Συγκεκριμένα, έκαναν 3 είδη αξιολογήσεων για κάθε αρχείο, κάθε setup και τις δύο μεθόδους εξαγωγής χαρακτηριστικών, χωρίς να γνωρίζουν ποιο clustering προέρχεται από κάθε μέθοδο:

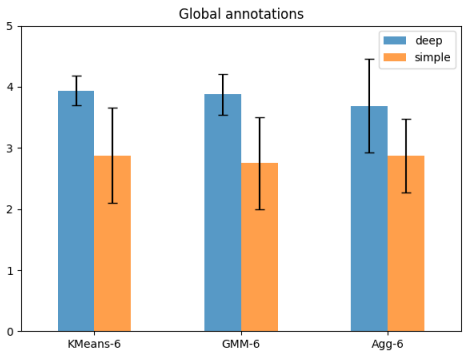
1. Global επισημειώσεις: Ο χρήστης βαθμολογεί τη συσταδοποίηση με έναν βαθμό από το 1 (κακό) έως και το 5 (καλό).

2. Cluster επισημειώσεις: Ο χρήστης βαθμολογεί κάθε cluster (ομάδα) με έναν βαθμό από το 1 (κακό) έως και το 5 (καλό).

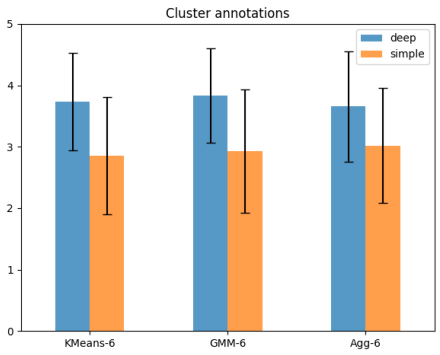
3. Επισημειώσεις σημείων: Ο χρήστης επιλέγει USVs από κάθε cluster και επισημαίνει αν καλώς ανήκουν στο συγκεκριμένο cluster ή αν θα έπρεπε να ανήκουν σε ένα άλλο.

Τα πιο χρήσιμα αποτελέσματα φαίνονται στα γραφήματα 0.5a και 0.5b, όπου φαίνονται οι μέσοι όροι των βαθμών για κάθε αρχείο ήχου και για κάθε χρήστη.

Όπως παρατηρούμε από τα αντίστοιχα διαγράμματα, η μέθοδος δεν παίζει ιδιαίτερο ρόλο, ενώ και στις δύο κατηγορίες επισημειώσεων, οι βαθμολογίες είναι υψηλότερες στις συσταδοποιήσεις που προέκυψαν από τα χαρακτηριστικά βαθιάς μάθησης. Επίσης, τρέξαμε και ένα



(a) Global επισημειώσεις: Μέση τιμή και τυπική απόκλιση των βαθμών που δόθηκαν σε κάθε setup, λαμβάνοντας υπόψη και τα 4 αρχεία ήχου, και τους 4 χρήστες.



(b) Cluster επισημειώσεις: Μέση τιμή και τυπική απόκλιση των βαθμών που δόθηκαν σε κάθε cluster από κάθε setup, λαμβάνοντας υπόψη και τα 4 αρχεία ήχου, και τους 4 χρήστες.

Figure 0.5: Global και cluster επισημειώσεις.

t-test που έδειξε υψηλή στατιστική σημαντικότητα των ευρημάτων και για τις δύο μορφές αξιολόγησης.

Επίσης, συγκρίναμε τα ποσοστά αποδοχής των USVs στα clusters τους, όπως φαίνεται στο γράφημα 0.6.

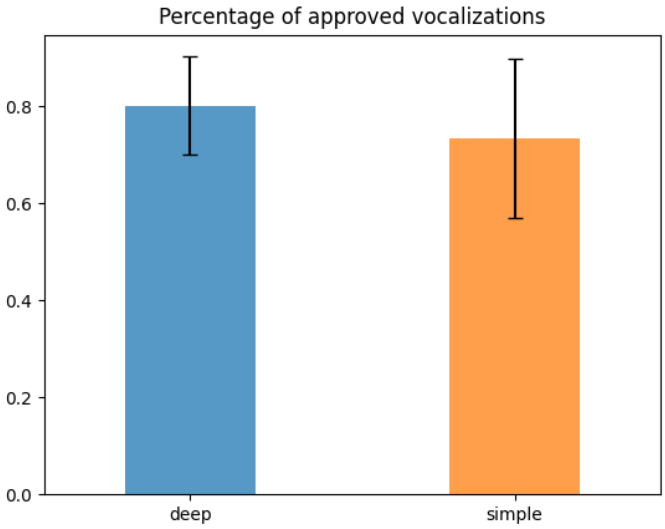


Figure 0.6: Μέσο ποσοστό αποδοχής των USVs στο cluster που έχουν ανατεθεί.

Είναι σαφές πως περισσότερα USVs έγιναν αποδεκτά στα clusters τους όταν το clustering έγινε με τα χαρακτηριστικά του autoencoder.

Ποσοτικοποιώντας αυτές τις παρατηρήσεις, συμπεραίνουμε ότι η βαθιά εξαγωγή χαρακτηριστικών υπερέχει της παραδοσιακής σε όλους τους τομείς:

- Οι βαθμοί των global επισημειώσεων είναι 37% υψηλότεροι για τη βαθιά προσέγγιση (Figure 0.5a)
- Οι βαθμοί των cluster επισημειώσεων είναι κατά μέσο όρο 30% υψηλότεροι για τη βαθιά προσέγγιση (Figure 0.5b)

- Το μέσο ποσοστό αποδοχής USVs ήταν 10% υψηλότερο για τη βαθιά προσέγγιση. (Figure 0.6)

0.6 Ημιεπιβλεπόμενη μάθηση για τη βελτίωση της συσταδοποίησης των USVs

Το επόμενο βήμα είναι η χρήση ημιεπιβλεπόμενης μάθησης για την πιθανή βελτίωση του clustering που πετύχαμε με την τελείως μη επιβλεπόμενη προσέγγιση.

0.6.1 SDEC

Για τον σκοπό αυτό χρησιμοποιείται η μέθοδος SDEC, στην οποία χρησιμοποιούμε έναν ήδη εκπαιδευμένο autoencoder (όπως στην περίπτωσή μας), και από τις εξόδους του encoder υπολογίζουμε μία αρχική συσταδοποίηση εφαρμόζοντας τον αλγόριθμο K-Means. Η μέθοδος αυτή προσπαθεί ταυτόχρονα να βελτιώσει της διαθέσιμες αναπαραστάσεις της εξόδου του encoder, δίνοντας έμφαση στο clustering και λαμβάνοντας υπόψη περιορισμούς που τίθενται στα πρότυπα εισόδου (USVs) σχετικά με το κατά πόσο θα θέλαμε να ανήκουν στο ίδιο cluster ή όχι.

Σε αυτό το πλαίσιο, η συνάρτηση κόστους ορίζεται τώρα όχι μόνο από το σφάλμα ανακατασκευής όπως στην τυπική εκπαίδευση του autoencoder, αλλά και από έναν όρο που επιδιώκει την ομογενοποίηση του clustering προσπαθώντας να πλησιάσει την κατανομή των δεδομένων μας (q) ως προς τα κέντρα του KMeans clustering που έχουν ήδη υπολογιστεί με μία κατανομή-στόχο (p)(απόκλιση Kullback Leibler), δίνοντας κατά τη διάρκεια της εκπαίδευσης μεγαλύτερη έμφαση στα παραδείγματα που ανατέθηκαν σε κάποια ομάδα με μεγάλη βεβαιότητα, και έναν όρο που εξασφαλίζει τους περιορισμούς στα ζευγάρια των USVs. Ο τελευταίος όρος έχει ως στόχο να επιβάλλει ποινή όταν δύο αναπαραστάσεις είναι κοντά, ενώ θα έπρεπε να ανήκουν σε διαφορετικές ομάδες, καθώς και όταν δύο αναπαραστάσεις είναι απομακρυσμένες, ενώ θα έπρεπε να ανήκουν στην ίδια ομάδα.

Τα ζευγάρια επιλέγονται σε κάθε batch του συνόλου εκπαίδευσης κατά την πρώτη εποχή και η πληροφορία για το αν πρέπει να ανήκουν στο ίδιο cluster ή όχι αποθηκεύεται σε έναν αρχικά μηδενικό τετραγωνικό πίνακα A, με μία θετική τιμή στο στοιχείο a_{ij} αν το ζευγάρι $i-j$ πρέπει να ανήκει στο ίδιο cluster και με μία αρνητική τιμή στο στοιχείο a_{ij} στην αντίθετη περίπτωση.

Επομένως, η συνάρτηση κόστους είναι τώρα η εξής:

$$L = \gamma_1 \cdot \text{BCELoss}(r, \hat{r}) + \gamma_2 \cdot KL(P\|Q) + \gamma_3 \cdot \frac{1}{N \cdot n_e} \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} \|z_i - z_j\|^2 \qquad (0.6)$$

όπου n_e είναι αριθμός της τρέχουσας εποχής. Τα βάρη γ_1, γ_2 και γ_3 επιλέγονται από τον χρήστη ανάλογα με την έμφαση που επιθυμεί να δώσει σε κάθε όρο. Η συνάρτηση αυτή χρησιμοποιείται τόσο για την εκπαίδευση του δικτύου, όσο και για την ανανέωση των κέντρων των συστάδων.

Αξίζει να σημειωθεί ότι η δική μας ομαδοποίηση δεν γίνεται στις αναπαραστάσεις εξόδου του encoder, εφόσον υφίστανται επεξεργασία πριν περάσουν στους αλγορίθμους συσταδοποίησης. Επομένως η συσταδοποίηση που προκύπτει με την παραπάνω διαδικασία δεν χρησιμοποιείται τελικά από εμάς, απλώς επιλέξαμε να την κρατήσουμε για να ενισχύσουμε την εκπαίδευση με προσανατολισμό στο clustering.

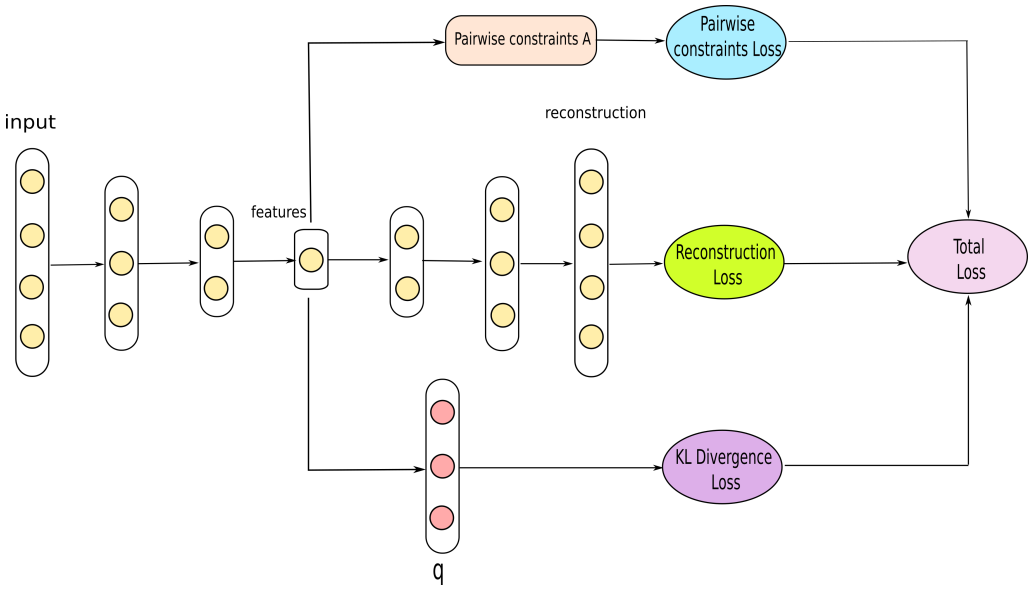


Figure 0.7: Η προσέγγισή μας στον αλγόριθμο SDEC.

0.6.2 Πειραματική αξιολόγηση της προσέγγισής μας στον αλγόριθμο SDEC

Για την αξιολόγηση χρησιμοποιήσαμε το σύνολο δεδομένων D3. Για κάθε ένα από τα 4 ηχογραφημένα αρχεία που το αποτελούν, επισημειώσαμε 1000 τυχαία επιλεγμένα ζευγάρια σημείων με την πληροφορία αν πρέπει να ανήκουν στο ίδιο cluster (ετικέτα 1) ή όχι (ετικέτα -1). Αυτές οι επισημειώσεις μπορούν να χρησιμοποιηθούν ως δεδομένα αλήθειας. Από το clustering, παίρνουμε τα αντίστοιχα ζευγάρια και τους δίνουμε αντίστοιχα την ετικέτα 1 αν έχουν τοποθετηθεί μαζί, και την ετικέτα -1 διαφορετικά. Έτσι, αυτή η αξιολόγηση αντιμετωπίζεται όπως σε ένα πρόβλημα ταξινόμησης.

Συγκρίθηκαν τα clusterings (agglomerative με 6 clusters) που φαίνονται στον πίνακα 0.3. Λόγω του γεγονότος ότι από τα δεδομένα αλήθειας, το πιο πιθανό ένα τυχαία επιλεγμένο ζευγάρι να έχει την ετικέτα -1, το πρόβλημα θεωρείται μη ισορροπημένο, επομένως κρίνεται σκόπιμη η χρήση της μετρικής macro F1 score.

Clustering approach	Mean Macro F1 (+/- std) (%)
Simple	61.75 (+/- 2.85)
Deep	63.25 (+/- 2.59)
Deep retrained (1 pair/batch)	65.30 (+/- 2.02)
Deep retrained (3 pairs/batch)	65.00 (+/- 1.57)
Deep retrained (5 pairs/batch)	64.95 (+/- 2.61)

Table 0.3: Μέση τιμή των macro F1 scores από τα 4 αρχεία ήχου. Στις περιπτώσεις που χρησιμοποιούνται βαθιά χαρακτηριστικά μετά από retraining, η τιμή αντιστοιχεί στον μέσο όρο των macro F1 scores που υπολογίστηκαν σε 5 διαφορετικά πειράματα (σε κάθε αρχείο ήχου).

Παρατηρούμε ότι υπάρχει βελτίωση της τάξης του 2% με την εφαρμογή της ημιεπιβλεπόμενης προσέγγισης σε σχέση με την πλήρως μη επιβλεπόμενη. Η αύξηση του αριθμού των ζευγαριών που επισημαίνονται με περιορισμούς δείχνει μικρή χειροτέρευση στο μέσο score, αλλά τα πειράματα δεν είναι αρκετά για να μπορούμε να βγάλουμε ασφαλή συμπεράσματα για τόσο μικρές διαφορές.

Είναι σαφές ότι αυτός ο τρόπος αξιολόγησης παρέχει μια γενική μόνο ιδέα της ποιότητας της συσταδοποίησης, καθώς λαμβάνουμε υπόψη ένα πολύ μικρό υποσύνολο των πιθανών ζευγαριών. Σε κάθε περίπτωση, αυτή η μέθοδος προσφέρει μια εναλλακτική πρόταση στο clustering των USVs ενός συγκεκριμένου αρχείου, με την καθοδήγηση ανθρώπινου παράγοντα, που ενδέχεται να είναι πιο κατάλληλη ανάλογα με την εφαρμογή.

0.6.3 Ταξινόμηση των ανιχνευθέντων USVs σε online εφαρμογές

Το βελτιωμένο clustering μπορεί να μας παρέχει δεδομένα, δηλαδή ζευγάρια αναπαραστάσεων των USVs με το αντίστοιχο cluster τους, τα οποία μπορούν να χρησιμοποιηθούν για την εκπαίδευση ενός ταξινομητή. Αυτός ο ταξινομητής μπορεί στη συνέχεια να χρησιμοποιηθεί σε εφαρμογές πραγματικού χρόνου, όπου μαζί με το χρονικό διάστημα κάθε USV θα παρέχεται στον χρήστη και η κλάση του. Αυτή η δυνατότητα θα επιτρέψει την εξέλιξη των πειραμάτων για τη συμπεριφορά των ποντικιών, καθώς αυξάνει τις διαθέσιμες πληροφορίες από τα ποντίκια σε πραγματικό χρόνο.

0.7 Συμπεράσματα και μελλοντικές προεκτάσεις

0.7.1 Συμπεράσματα

Συνολικά, αναπτύξαμε ένα εργαλείο για ανίχνευση και κατηγοριοποίηση USVs, που χαρακτηρίζεται από τις εξής καινοτομίες:

- **Offline ανίχνευση των USVs**. Η μεθοδολογία που αναπτύξαμε είχε πολύ καλή επίδοση και σε καθαρά και σε θορυβώδη αρχεία ήχου, όπως προέκυψε από τη σύγκριση με άλλες μεθόδους. Η παραμετροποίησιμη φύση της επιτρέπει αλλαγές στα κριτήρια κατωφλίωσης, επιτρέποντας την προσαρμογή της σε ποικιλία συνθηκών ηχογράφησης και καθιστώντας την ευέλικτη και αξιόπιστη.

- **Ανίχνευση των USVs σε πραγματικό χρόνο**. Η δυνατότητα που παρέχεται για χρήση του εργαλείου μας σε εφαρμογές πραγματικού χρόνου, με επίδοση ανίχνευσης ισάξια με τις online μεθόδους είναι αρκετά πρωτοπόρα, καθώς η παροχή άμεσης πληροφορίας για τα παραγόμενα USVs από τα ποντίκια μπορεί να βοηθήσει στη μελέτη των αντιδράσεών τους σε συγκεκριμένα ερεθίσματα.

- **Βαθιά εξαγωγή χαρακτηριστικών και συσταδοποίηση**. Χρησιμοποιήσαμε έναν convolutional autoencoder για την εξαγωγή χαρακτηριστικών από εικόνες με μη επιβλεπόμενο τρόπο και στη συνέχεια προχωρήσαμε σε συσταδοποίηση με βάση αυτά. Όπως προέκυψε από την πειραματική αξιολόγηση, η επίδοση της συσταδοποίησης ήταν καλύτερη όταν χρησιμοποιήθηκαν αυτά τα χαρακτηριστικά και όχι αυτά που προέκυψαν από κλασικές τεχνικές επεξεργασίας σημάτων. Ο μη επιβλεπόμενος χαρακτήρας τόσο στην εξαγωγή των χαρακτηριστικών, όσο και στην κατηγοριοποίηση παρέχει στην όλη διαδικασία μία ελευθερία από τους περιορισμούς των προκαθορισμένων κλάσεων ή χαρακτηριστικών, δίνοντας τη δυνατότητα νέων, ομογενών ομαδοποιήσεων με όχι τόσο τυπικά χαρακτηριστικά.

- **Ημιεπιβλεπόμενη προσέγγιση της συσταδοποίησης**. Η ενίσχυση της ποιότητας της συσταδοποίησης μπορεί να γίνει με χρήση καθοδήγησης από τον χρήστη, ο οποίος παρέχει πληροφορία για το αν ένα ζευγάρι USVs θα έπρεπε να ανήκει στην ίδια ομάδα ή όχι. Με βάση την πειραματική αξιολόγηση, παρατηρήσαμε όντως βελτίωση σε σχέση με τη μη επιβλεπόμενη διαδικασία. Αυτή η λειτουργία του AMVOC μπορεί να αποδειχθεί χρήσιμη για διαφοροποιήσεις της συσταδοποίησης που να εξυπηρετούν καλύτερα τις ανάγκες του χρήστη ή μιας εφαρμογής.

- **Ταξινόμηση των USVs σε πραγματικό χρόνο**. Η βελτιωμένη συσταδοποίηση μπορεί να παρέχει δεδομένα εκπαίδευσης ενός ταξινομητή, ο οποίος θα χρησιμοποείται για την ταξινόμηση USVs σε εφαρμογές πραγματικού χρόνου. Αυτή η δυνατότητα είναι πραγματικά καινοτόμα, καθώς ο χρήστης θα έχει πληροφορία για το πότε παρήγαγε κάποιο USV το ποντίκι, αλλά και τι τύπου είναι, ανοίγοντας τον δρόμο για περαιτέρω πειράματα συσχέτισης της συμπεριφοράς των ποντικιών με συγκεκριμένους τύπους USVs και στην ανάπτυξη συμπεριφορικών εκτιμήσεων και αξιολογήσεων κλειστού βρόχου (με ανατροφοδότηση).

0.8 Μελλοντικές προεκτάσεις

Συνολικά, έχουμε αναπτύξει ένα ολοκληρωμένο εργαλείο που ελπίζουμε να εμπνεύσει νέες μελλοντικές κατευθύνσεις και να δώσει μια πιο σαφή εικόνα για τη συμπεριφορά και την επικοινωνία των ποντικιών.

Αρχικά, θα μπορούσαν να εξερευνηθούν εναλλακτικές επιλογές για τη μη επιβλεπόμενη εξαγωγή χαρακτηριστικών. Αντί του κλασικού convolutional autoencoder, θα μπορούσαμε να χρησιμοποίησουμε μια παραλλαγή, όπως για παράδειγμα τον denoising autoencoder, ο οποίος δεν μαθαίνει μέσα από την ανακασκευή του προτύπου εισόδου, αλλά προσπαθώντας να αφαιρέσει θόρυβο από αυτά.

Μια τελείως διαφορετική προσέγγιση θα ήταν η αλλαγή μοντέλου, όπως η χρήση ενός GAN (Generative Adversarial Network). Τα GANs αποτελούνται από έναν generator (που προσπαθεί να παράγει δείγματα όπως τα πρότυπα εισόδου) και έναν discriminator (που είναι ένα δίκτυο που λειτουργεί ως ταξινομητής των δειγμάτων σε αυτά που ανήκουν όντως στο σύνολο δεδομένων εκπαίδευσης και σε αυτά που έχουν παραχθεί από τον generator). Ο discriminator μέσα από αυτή τη διαδικασία μαθαίνει χαρακτηριστικά των δειγμάτων εκπαίδευσης και αν πάρουμε τις αναπαραστάσεις πριν την κεφαλή ταξινόμησης, θα έχουμε διανύσματα χαρακτηριστικών. Η σύγκριση της συσταδοποίησης με αυτά σε σχέση με τα χαρακτηριστικά που εξάγαμε εμείς θα παρουσίαζε ενδιαφέρον.

Όσον αφορά την ημιεπιβλεπόμενη προσέγγισή μας, αυτή μπορεί να μελετηθεί περαιτέρω, ιδιαίτερα ως προς τον ρόλο που παίζουν τα ζευγάρια που επισημειώνει ο χρήστης κατά τη διαδικασία της εκπαίδευσης στο συνολικό clustering. Η ανακάλυψη κάποιου μοτίβου στην επιλογή των ζευγαριών που να εξασφαλίζει βελτίωση της ποιότητας του clustering παρουσιάζει ιδιαίτερο ενδιαφέρον.

Επιπλέον, θα μπορούσαν να δοκιμαστούν διαφορετικές πιθανές μεθοδολογίες για ημιεπιβλεπόμενη μάθηση με άλλου τύπου παρέμβαση του ανθρώπινου παράγοντα στη διαδικασία εκπαίδευσης. Η σύγκριση τέτοιων μεθόδων με τη δική μας υλοποίηση των περιορισμών ανά ζευγάρι USVs θα ήταν επίσης ενδιαφέρουσα.

Μία άλλη πρόκληση θα ήταν και η πιθανή εύρεση άλλου τρόπου αξιολόγησης της συσταδοποίησης των USVs, που θα ήταν λιγότερο υποκειμενικός και περισσότερο αυτοματοποιημένος.

Επίσης, θεωρούμε πως πολλά υποσχόμενη θα ήταν η εξερεύνηση πιθανής χρονικής και ακολουθιακής συσχέτισης μεταξυ των διαδοχικών USVs. Η ανακάλυψη συγκεκριμένων μοτίβων τόσο σε μικρές όσο και σε μεγαλύτερες, χρονικά, κλίμακες θα επέτρεπε τη σύνδεση ακολουθιών από USVs με ποικίλες συμπεριφορές των ποντικιών και μια γενικότερη κατανόηση της χρήσης της φωνητικής τους δραστηριότητας.

Αυτός ο στόχος θα μπορούσε να επιτευχθεί και με την ενδεχόμενη χρήση του πλαισίου (context) της ηχογράφησης στον αλγόριθμο κατηγοριοποίησης. Αυτό θα γινόταν με τη μορφή metadata που θα αφορούσαν π.χ. συμπεριφορές του ποντικιού ή ερεθίσματα με τα οποία ήρθε σε επαφή την ώρα της ηχογράφησης και αποτελεί πρόκληση τόσο από άποψη εξελικτικής βιολογίας, όσο και μηχανικής μάθησης.

Chapter 1

Introduction

This chapter's goal is to describe the motivation behind this work and introduce readers to the actual problem. It also contains previous relevant work, as well as a brief outline of the rest of the thesis and a description of the data sets used for experiments and tuning throughout our work.

1.1 Motivation

Vocalizations play a significant role in social communication across species (Bradbury and Vehrencamp (2011)). It has been discovered that rodents and bats use vocal communication to exchange information about their current states. This information can give us insight into mice behavior and actions and help us understand the way mice function. More specifically, it can possibly be used for the identification of individual mice or groups, structure of a group (e.g. dominance) and subsequent feelings (e.g. fear, aggression, competitiveness), next behavior (e.g. play, attack, approach), environment conditions (e.g. presence of food or enemies/predators) and mother-pup interactions (Hoffmann et al. (2012); Nyby et al. (1976); Neunuebel et al. (2015); Slobodchikoff et al. (2012); D'Amato et al. (2006)). This study could then lead in a better understanding of the neural basis of vocalizations (underlying neurobiological processes) and the role of genes in neuroanatomy, by studying impact of mice genes on their vocal development ((Grimsley et al. 2011b; Bowers et al. 2013; Chabout et al. 2016; Tabler et al. 2017)). Neural mechanisms underlying USVs are a useful model for the neurobiology of human speech and speech-related disorders (often genetically caused) (Ferhat et al. (2016)).

More specifically, mice emit ultrasonic vocalizations (USVs, 30-110 kHz) relative to the human hearing range (2-20 kHz), meaning that humans cannot hear them. These vocalizations can be complex and can contain multiple *syllables*, which are defined as continuous units of vocal sound not interrupted by a silence period and form sequences(Arriaga et al. (2012); Holy and Guo (2005)). There are many types of syllables generated by mice; all syllable types used by a specific mouse in a certain condition comprise a syllable repertoire (Van Segbroeck et al. (2017)). In general, transitions between syllables, and possible differences in acoustic structure (e.g. mean frequency, amplitude) vary according to behavioral and social environments, genetic strain (such that USVs can be used as a phenotyping marker for different genotypes) and development stage((Chabout et al. (2015); Fonseca et al. (2021); Grimsley et al. (2011a); Melotti et al. (2021))). Understanding complex vocalization structure of mice will be crucial to advancing vocal and social communication research.

Exploring the social meaning of USVs has been an interesting task during the past years; however, although there is strong evidence that they actually play an important role in mice communication, it is still not clear which characteristics and syllable types are related

to certain biological states or social conditions. To assess the social meaning of USVs, it is necessary to first categorize different syllables and then probably correlate each class to specific behavioral patterns. Up to now, multiple tools have been developed whose aim is to firstly detect USVs in an audio recording and then classify them according to specific features.

USV detection has been especially challenging, since tools are often developed taking into account certain recording setups and are not always capable of generalizing. Recently, new tools have employed machine learning and neural networks for detection, which has increased detection accuracy, though they are quite dependent on data used for their training and generalizing isn't necessarily guaranteed. Another concern with these methods is their detection speed, since involving deep networks in the detection procedure might indicate considerable latency. An important detail is that in general, developed tools process mice recordings off-line; though it would be extremely useful to be able to get real-time feedback of detected USVs and observe the mouse behavior simultaneously, for example how they respond (vocally) to a specific stimulus.

On the other hand, classification of detected syllables is also tricky. There have been both supervised and unsupervised approaches. In supervised methods, there are some manually predefined classes and each syllable is categorized in one of these. In this case, it is usual to use a neural network which will be trained to classify syllable types. Unsupervised methods are trying to group vocalizations by some of their features. In this case, it is most common that hand-crafted (human defined) features are calculated for each syllable and then syllables are organized in groups (clusters). In both cases, human intervention in the procedure (either by setting specific classes or defining decisive features) might limit the exploration possibilities of the tool. In fact, there is no conclusion on which classification schemes or syllables features provide the best biological insights (Van Segbroeck et al. (2017)), since features may differentiate across varying conditions.

1.2 Related Work

A broad interest in analyzing behavior and phenotyping with USVs has led to a proliferation of tools for the detection of mouse vocalizations. Mouse Ultrasonic Profile ExTraction (MUPET) is a MATLAB open source tool, developed by Van Segbroeck et al. (2017) to detect syllables, analyze the vocalizations features and cluster the syllables depending on these features. MUPET first filters the signal to keep high frequencies (25-125 kHz). It then uses spectral subtraction to remove stationary noise, and at last it computes the power of the spectral energy in the ultrasonic range above a specific threshold. The vocalizations are converted to representations by using negative matrix factorization (NMF) and gammatone filters. The filtered spectrograms are then used to cluster vocalizations based on spectral shape similarities. The clustering is done by using K-Means, and user-defined number of "repertoire units" (clusters). The authors note that MUPET can also be used with many non-rodent species' vocalizations.

DeepSqueak is a software suite for USVs detection and analysis (Coffey et al. (2019)). It splits the recording into areas, computes the corresponding sonograms and passes them to a Faster-RCNN (recurrent convolutional Neural Network) object detector, which consists of two networks. The first network is a region detection network, which proposes sections of the spectrogram that could contain actual vocalizations. These sections are then used as inputs to a second network, a convolutional neural network (CNN), and are classified depending on whether or not the sections contain vocalizations. This process has been recently updated to use a You Only Look Once (YOLO) network. DeepSqueak can also be used for clustering the detected syllables, either with a supervised or unsupervised method. Their unsupervised approach gives the user the opportunity to define three weighted input

features: shape, frequency and duration of the vocalization. An important difference from MUPET is that the clustering of MUPET takes syllable amplitude into account, whereas DeepSqueak does not, which can be considered an advantage, given that the volume of a vocalization can also depend on the recording setup, among other factors. DeepSqueak's networks can also be trained on new vocalizations, meaning it can potentially be improved for specific experimental needs.

VocalMat is a MATLAB tool (Fonseca et al. (2021)), which uses image processing techniques and differential geometry analysis on the spectrogram of a recording to detect vocalization candidates. It can then classify detected USVs into 12 predefined categories (including noise), by using a CNN. As with DeepSqueak, VocalMat's networks can be retrained as well.

A more recent tool is Deep Song Segmenter (DeepSS), which has been used for annotation of songs of mice, birds and flies (Steinfath et al. 2021). DeepSS learns a representation of sounds features directly from raw audio recordings using temporal convolutional networks (TCNs), based on dilated convolutions. It is a comparatively fast, supervised annotation method, since the network is trained with manually annotated recordings. It can also be combined with unsupervised approaches to reduce the amount of manual annotation required.

Other tools were developed for mouse USV detection that rely more explicitly on the acoustic parameters of the recordings and do not use machine learning methods or clustering techniques to classify detected USVs. One such tool is Mouse Song Analyzer (Holy and Guo 2005; Arriaga et al. 2012; Chabout et al. 2015), which has recently been rewritten and improved from the original MATLAB implementation to a Python implementation with added filtering components to improve USV detection rates (cite Peter DOI), hereafter referred to as MSA1 and MSA2, respectively. MSA1 first generates a spectrogram from the audio recording, and subsequently thresholds it to remove white noise. Frequencies outside of mouse USV song range (35-125 kHz) are discarded. USVs are detected in the spectrogram are determined by surpassing a combination of user-defined thresholds for frequency, amplitude, spectral purity, and duration. MSA2 first bandpasses the raw audio (30-115 KHz) and then generates a spectrogram. The spectrogram is thresholded according to the signal-to-noise ratio at each frequency. In both MSA1 and MSA2, USVs are detected in the spectrogram by surpassing a combination of user-defined thresholds for frequency, amplitude, spectral purity, and duration. The detected USVs are then classified based on the number of gaps, or "pitch jumps", that are present within the detected USVs (Chabout et al. 2015).

Another tool is Ax (Neunuebel et al. (2015)), which tries to detect vocal signals by keeping time-frequency points of the spectrogram that signifcantly exceed noise values. They mainly focus on matching the recorded vocalizations to the specific mouse that produced it (when there are multiple mice in the cage) and also correlating the produced vocalization with a certain social behavior of the mouse. USVSEG (Tachibana et al. (2020)) is also a MATLAB tool used for vocalization detection, emphasizing noise removal of the spectrogram in the cepstral domain before thresholding to detect whether a segment contains a vocalization or not.

1.3 Research objective

Taking into consideration the problems and difficulties described in Section 1.1, our goal is to create a mice vocalizations tool, which will be accurate, efficient and easy-to-use, both for USVs detection and for their categorization and grouping.

More specifically, we have developed a tool called AMVOC (Analysis of Mouse VOcal Communication) for mice USVs research. The functionality of AMVOC is twofold:

1. *USVs detection* in audio recordings, by applying dynamical spectral thresholding similar to MSA, both in off- and online mode

2. *Clustering* of detected USVs based on a set of features, which are extracted in completely unsupervised manner from a deep convolutional autoencoder. There is also the opportunity to enhance clustering performance by a semisupervised approach used for clustering refinement.

Our proposed method of detection has been extensively evaluated using real recordings and compared to plenty of the methods presented in Section 1.2, being shown to outperform most of these tools in various acoustic environments and having a pretty consistent performance also in cases of high noise presence. This indicates that AMVOC can be used for processing of multiple recording set-ups, providing a flexible and reliable option. A major contribution in this field is our online vocalization detection method, which performs equivalently well to offline methods and also provides real-time feedback on a less than 1 second basis.

The deep feature extraction procedure, using a deep convolutional autoencoder, allows the exploration of high and low level features, which are used for clustering, that can possibly reveal biologically relevant USV clusters. This hypothesis is further supported by the fact that both feature extraction and clustering procedure are totally unsupervised, supplying the method with a freedom to discover characteristics as well as patterns that aren't manually predefined and specified.

In this case, human intervention can come in the form of providing the feature extraction and clustering algorithm with constraints regarding possible clusters, which can boost clustering performance depending on the application. Our proposed semisupervised approach still guarantees clustering freedom, while providing it with a helpful direction.

These improved clusterings (USV and corresponding cluster) can be used as ground truth training data of a classifier, which can then provide the classes of real-time detected USVs, further expanding experimentation opportunities.

In general, the combination of these functionalities and features offer a complete mice recordings processing tool, which can indeed help to better understand and analyze mice vocal behavior and connect it with social and environmental conditions.

1.4 Thesis outline

Chapter 2 presents the full background, which is helpful for completely understanding the concept of our ideas and implementations in the following chapters. It involves the basics of audio signal processing, but the main part of this chapter is undoubtedly machine learning and pattern recognition. These are analyzed quite extensively, but emphasis is put on the background of models and procedures we use in our methods, and more specifically deep neural networks (mostly convolutional neural networks), clustering methods and semisupervised deep embedded clustering, as well as useful evaluation metrics.

Chapter 3 focuses on the vocalizations detection methods, firstly on offline and then online processes. Then, a results section follows, presenting the experimental comparisons and evaluation of our detection methods.

Chapter 4 describes the deep feature extraction procedure by thoroughly going through model training and parameter tuning. A baseline feature extraction procedure (giving a set of hand-crafted features per vocalization) is also presented. Clusterings are produced using both feature types and the two approaches are compared and evaluated using human annotations and ratings.

Chapter 5 emphasizes the semisupervised extension of the method presented in chapter 4 and the new additions introduced to the typical unsupervised approach, along with a

results section, which basically compares the semisupervised to the unsupervised outcomes. It also refers to the training of a classifier using clustering data, and its subsequent use for real-time USVs classification.

Chapter 6 is the final chapter, summarizing the conclusions drawn from the total and the novel contributions of our work.

1.5 Datasets

In the course of our experimental and evaluation procedures, we created three datasets (D1, D2 and D3).

Dataset D1 was created to evaluate and compare vocalization detection methods. More specifically, we compiled a ground-truth dataset of 9 audio segments of 5-10 seconds each, containing 245 syllables in total. The ground-truth annotation was performed by a domain expert by simply declaring the frames that correspond to actual vocalizations, with a time resolution of 1 ms.

Dataset D2 consists of 26 different recordings, used as the training set of our convolutional autoencoder.

Dataset D3 was created for the experimental evaluation of the clustering setups. We used a dataset of 72 behavioral recordings, 36 in the category FemaleUrineDirected and 36 in the category FemaleLiveDirected (see Methods for details). We have randomly selected 20 s from each recording, where the vocalization rate should be at least 2.5 vocalizations/sec. We then concatenated the 20 seconds interval from each recording to a new recording. We generated 4 recordings, 2 from the FemaleUrineDirected category and 2 from the FemaleLiveDirected category.

Chapter 2

Background

In this section we are going to present background material, essential for the explanation and full understanding of the methods we have used in the main part of the thesis. Firstly, we will go through some audio signal processing basics, and then the major background part regarding pattern recognition and machine learning.

2.1 Audio Signal processing

Signal is a term used to describe something that carries some sort of information, for example about the state or behavior of a physical system. Most commonly, though, signals refer to sound, images or videos.

Signals contain information in a pattern of variations; thus, they are represented as mathematical functions. A sound signal, for example, is represented as a function of time. In general, sound is an analog signal, which is basically described by two main characteristics: *amplitude*, which shows how loud the sound is, and *frequency*, which shows how fast the wave vibrations of sound are.

In order for a sound signal to be represented on computer, it needs to be converted in digital form. The analog to digital conversion consists of two important operations: sampling and quantization.

Sampling is the operation of converting a continuous-time signal $x_c(t)$, $t \in \mathbb{R}$ to a discrete-time signal $x[n]$, $n \in \mathbb{Z}$. Continuous-time signals' independent variable is continuous, whereas discrete-time signals are defined at discrete times, so they are described by a sequence of numbers. To convert a continuous-time signal to a discrete-time signal, we sample the continuous signal every T_s time units, where T_s is referred to as sampling period. We also often refer to the sampling frequency as $f_s = \frac{1}{T_s}$. It is obvious that $x[n] = x_c(nT_s)$.

Quantization is the operation of converting a continuous-amplitude signal to a discrete-amplitude signal. This means that we want the amplitude of the signal to take specific predefined (quantized) values, which depend on the quantizer we use. Then, every value of the input signal $x[n]$ is replaced by its closest one of the set of quantized values: $\tilde{x}[n] = Q(x[n])$.

Signals for which both time and amplitude are discrete are *digital signals*.

A very useful and commonly used signal is the *unit sample sequence* $\delta[n]$, defined as:

$$\delta[n] = \begin{cases} 0, & n \neq 0 \\ 1, & n = 0 \end{cases} \tag{2.1}$$

An interesting property is that every signal $x[n]$ can be written as the weighted sum of shifted unit sample sequences:

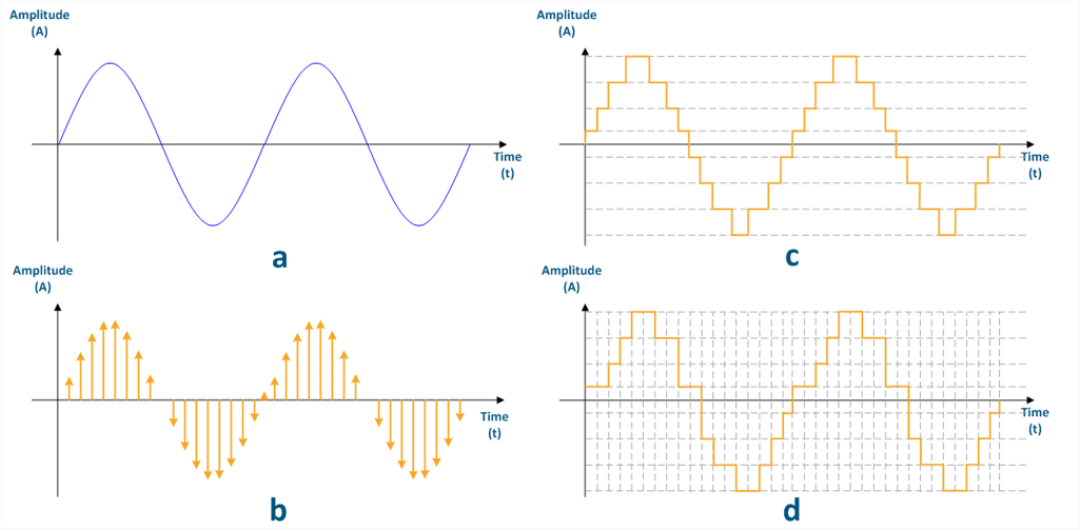


Figure 2.1: Different signal types. a) Analog signal (continuous time and amplitude). b) Discrete-time and continuous-amplitude signal. c) Continuous-time and discrete-amplitude signal. d) Digital signal (discrete time and amplitude).

$$x[n] = \sum_{k=-\infty}^{+\infty} x[k]\delta[n-k] \tag{2.2}$$

2.1.1 Discrete Time Systems

A system is defined as an operation or transformation that maps an input sequence $x[n]$ into an output sequence $y[n]$:

$$y[n] = T\{x[n]\} \tag{2.3}$$

and is often represented as in Figure 2.2.

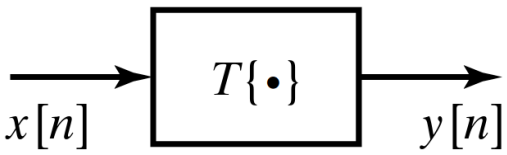


Figure 2.2: A discrete-time system.(Oppenheim and Schafer (2009))

A very important type of systems is the LTI (linear time invariant) systems, since they combine two important properties: linearity and time invariance. Linearity is practically equivalent to superposition. If y_1 is the output of the system when the input is x_1 and y_2 the output when the input is x_2, then a linear system has the following properties:

$$T\{x_1[n] + x_2[n]\} = T\{x_1\} + T\{x_2\} = y_1[n] + y_2[n] \tag{2.4}$$

and

$$T\{ax[n]\} = aT\{x[n]\} = ay_[n] \tag{2.5}$$

where $a \in \mathbb{R}$.

Time invariance means that if the input sequence has a time shift or delay, then the same shift is caused by the system in the output sequence. More specifically, if an input sequence $x[n]$ results in an output sequence $y[n]$, then for every n_0 the input sequence $x_1[n] = x[n - n_0]$ will result in output sequence $y_1[n] = y[n - n_0]$.

LTI systems have a significant property: they are fully defined by their impulse response. More specifically, if $h_k[n]$ denotes the response of the system for the input sequence $\delta[n-k]$ then:

$$y[n] = T\{x[n]\} = T\{\sum_{k=-\infty}^{+\infty} x[k]\delta[n-k]\} \tag{2.6}$$

Using the linearity property:

$$y[n] = \sum_{k=-\infty}^{+\infty} x[k]T\{\delta[n-k]\} = \sum_{k=-\infty}^{+\infty} x[k]h_k[n] \tag{2.7}$$

Because of the time invariance property, we have:

$$y[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n-k] \tag{2.8}$$

The operation defined above is called *convolution* and is represented by the following notion:

$$y[n] = x[n] * h[n] \tag{2.9}$$

It can be proven that

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n-k] = \sum_{k=-\infty}^{+\infty} x[n-k]h[k] \tag{2.10}$$

This means that the output of any Linear Time Invariant system can be calculated using the input sequence and the impulse response.

LTI systems are often used for filtering a signal. For example, a common smoothing filter which reduces the highest amplitude values is the moving average filter. This filter replaces every value of the input sequence with the average of its M neighboring values. Its impulse response is $h[n] = \frac{1}{M}$. So, the output sequence is:

$$y[n] = \frac{1}{M}\sum_{k=0}^{M-1} x[n-k] \tag{2.11}$$

2.1.2 Fourier Transform

If the input sequence is $x[n] = \mathrm{e}^{j\omega n}$, then:

$$y[n] = \sum_{k=-\infty}^{+\infty} x[n-k]h[k] = \sum_{k=-\infty}^{+\infty} h[k]\mathrm{e}^{j\omega(n-k)} = H(\mathrm{e}^{j\omega})\mathrm{e}^{j\omega n} \tag{2.12}$$

where $H(\mathrm{e}^{j\omega}) = \sum_{k=-\infty}^{+\infty} h[k]\mathrm{e}^{j\omega - k}$ is the *frequency response* of the system. This means that $\mathrm{e}^{j\omega n}$ is eigenfunction of the system and $H(\mathrm{e}^{j\omega})$ are the eigenvalues.

This is a fundamental property of LTI systems, which has led to signal representations as the weighted sum of complex exponentials or sinusoidals of different frequencies. Intuitively,

we want to determine the amplitude of the sinusoidal components that make up the signal. This amplitude is, in fact, a function of frequency.

This representation is achieved through the well known *Fourier Transform*, which has different forms, depending on whether the signal is continuous- or discrete-time. In discrete-time signals, we use the *Discrete Fourier Transform* (DFT) to define the weights or coefficients of the sinusoidals. For a finite length signal $x[n]$ with N samples, DFT is the result of sampling of:

$$X(\mathrm{e}^{j\omega}) = \sum_{n=0}^{N-1} x[n]\mathrm{e}^{-j\omega n} \tag{2.13}$$

The sampling step of the coefficients is equal to $\frac{2\pi k}{N}$. So, if we keep N coefficients, the discrete-frequency transform is defined by:

$$X[k] = \begin{cases} \sum_{n=0}^{N-1} x[n]\mathrm{e}^{-j\frac{2\pi k}{N}n}, & 0 \leq k \leq N-1 \\ 0, & \text{elsewhere} \end{cases} \tag{2.14}$$

and the signal can be written as:

$$x[n] = \begin{cases} \frac{1}{N}\sum_{k=0}^{N-1} X[k]\mathrm{e}^{j\frac{2\pi k}{N}n}, & 0 \leq n \leq N-1 \\ 0, & \text{elsewhere} \end{cases} \tag{2.15}$$

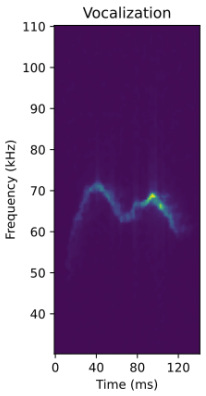
The coefficients $X[k]$ describe the signal in frequency domain and have many interesting properties. For example, the absolute amplitude of $X(\mathrm{e}^{j\omega})$ informs us about the power of the signal in frequency ω. This is also called the magnitude of the *frequency spectrum*. DFT is efficiently calculated with an algorithm named *Fast Fourier Transform* (FFT).

2.1.3 Spectrogram

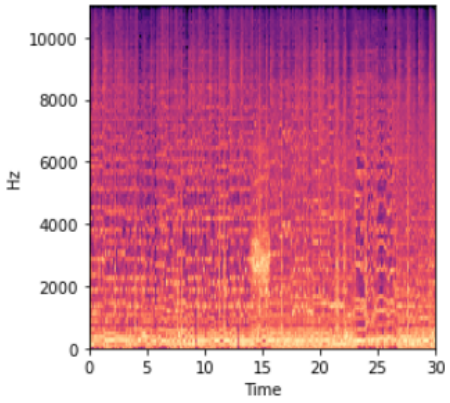
Using the Fourier Transform, we can get and process the frequency spectrum of a signal. However, the properties and characteristics of a signal are non-stationary; thus, we usually want to get an insight into the spectrum of the signal as it evolves with time. That's why we use *Short Term Fourier Transform*, which means that we split the signal into successive frames (that can be overlapping) and apply FFT to each frame. So, we have a frequency spectrum consisting of coefficients $X[k]$ for each frame.

We can use the resulting sequences to form an image, where the horizontal axis corresponds to time and the vertical to frequency. For each frame and since the magnitude spectrum is a function of frequency, it is vertically laid to span all the frequency bins of the vertical axis. So, the image consists of magnitudes of spectrums laid vertically side by side. Every resulting value indicates the energy of the signal in the specific time frame and the specific frequency. This image is called *spectrogram* and is very useful since it conveys both temporal and spectral information.

What's really interesting is that with a spectrogram we get the chance to explore and derive features of an acoustic signal (originally represented as an 1D array) with an image (represented as a 2D array with values in the interval [0,1] or [0,255]). (Oppenheim and Schafer (2009))



(a) Spectrogram of a mouse vocalization.



(b) Spectrogram of a recorded signal of classical music.

Figure 2.3: Spectrograms of different signals. It is interesting to note the different frequency range, apart from their obvious dissimilar shapes.

2.2 Introduction to Machine Learning and Pattern Recognition

The invention of thinking machines has been an idea of the human kind since ancient years (Talos, the giant bronze automaton of Greek mythology, is one of the first references to a robot in human history). However, the conception that a machine can be as intelligent as humans and even outperform them at specific tasks became realistic hundreds of years later, when first computers emerged and revolutionized every aspect of people's lives.

In our days, especially during the last decade, the main focus is on *Artificial Intelligence*, whose practical applications and current research topics are increasing and expanding. In the first steps of artificial intelligence, its goal was to solve problems which were challenging and intellectually hard for human beings, like complex mathematical operations, with impressive speed and accuracy. What's proven really hard though, is to solve problems that are easy, like automatic, for human brain, and that they can't be formally described, such as recognizing similar faces in images or spoken words. (Goodfellow et al. (2016))

In order for this kind of problems to be solved, machines need to learn from experience and examples. The procedure of running algorithms, that improve the ability of a computer to solve a specific problem through experience and by the use of data is called *Machine Learning*.

Pattern Recognition is the scientific field whose goal is the classification of observed objects into a number of categories or classes. These objects can be anything that needs to be classified, from images to signal waveforms and are described by the general term *patterns*.

Practical examples and applications of pattern recognition are many and are connected to multiple other disciplines and fields.

- *Computer vision* is an area where pattern recognition plays an important role. Computer vision systems take images and then try to analyze them. This analysis can include their classification to a number of different categories, or the detection and recognition of several objects depicted.

- *Character recognition* is another important field of pattern recognition that includes the recognition of different written characters, such as letters, numbers and other symbols. Furthermore, it can be used for handwriting and signature identification.

- *Computer-aided diagnosis* is a newly emerging application of pattern recognition, whose aim is to assist doctors at medical diagnoses. It mostly involves classification of measurements of patients into two categories: measurements that indicate a healthy or a problematic condition. These measurements can be scans (X-rays, computed tomographic images, ultrasound images), ECGs, EEGs etc.

- *Speech recognition* is another area of interest, which is significantly developed during the last years. It is associated with recognizing and understanding spoken information. The most common application is the communication between humans and machines via speech.

- *Data mining and information retrieval* from databases is also extremely popular, since it provides useful information in a wide range of fields, like medicine, financial analysis, image and music retrieval.

Of course, these are only basic examples that indicate how wide the range of applications of pattern recognition actually is. (Theodoridis and Koutroumbas (2009))

In order for the different patterns to be classified, the model or the algorithm we use must get the information that best describes every pattern. Therefore, each pattern is described by a *representation*, which carries the most important information about the pattern; information that makes the patterns distinct from one another. This representation consists of measurable quantities, known as *features*. In general, if l features x_i, $i = 1, ..., l$ are used, the representation is basically the *feature vector*

$$\mathbf{x} = [x_1, x_2, ..., x_l]^T \tag{2.16}$$

where T denotes transpose vector. Every feature vector is a unique identifier of a specific pattern. (Theodoridis and Koutroumbas (2009))

As mentioned earlier, the goal of pattern recognition is to successfully classify patterns to different categories. This can be achieved by using a model (classifier), which has to be trained with a machine learning algorithm, so that it can solve our specific classification problem. For this purpose, we need a data set of patterns that will be used for the *training* of the model. The patterns (feature vectors), whose real class is known and which are used for the training of the classifier are called *training feature vectors*. So, each feature vector is connected to an a-priori known label, which determines its class. The basic issues that arise when we want to design a classification system are in the same time steps of the design procedure pipeline. They are described below:

1. How are the features generated? The answer to this question is problem-dependent and corresponds to the *feature generation stage* of the design.

2. How do we choose how many and which specifically are the appropriate features that will make patterns distinct? This concerns the *feature selection stage* and is very important because it actually defines which features are the most important for the classification task. In fact, multiple features are generated and a subset of them is then chosen to be included in the feature vector.

3. Is the current space of the selected features one that enables an easy solution of the classification problem? Feature selection is important, though sometimes the selected features might need some kind of transformation in another space, that will probably make the classification problem easier to solve. This is called *feature pre-processing stage*. This procedure can also speed up computations, because it often employs dimensionality reduction of feature vectors.

Figure 2.4: Classification system design pipeline.

4. How is the classifier designed? This is the *classifier design stage*, which includes decisions such as which decision region to generate in the l-dimensional features space, that will successfully and accurately discriminate the different classes, or what the best criterion (loss function) that we will optimize is, to ensure the model is trained. Basically, this stage consists of two parts: the first one is to choose the general form of model and the second one is to use training samples to learn or estimate the parameters of the model that are more suitable for our specific problem and data set.

5. How can we measure the performance of the classifier, after its training is finished (*system evaluation stage*)? For this purpose, it is most common to use some feature vectors as input to the model and compare the label predicted from the model to the actual one. This data set of feature vectors is called *test set* and is used only for the evaluation of the already trained model. The ability to categorize correctly new examples that differ from those used for training is known as *generalization*.

(Theodoridis and Koutroumbas (2009), Duda et al. (2001), Bishop (2006)) The system design pipeline is displayed in Figure 2.4.

In general, any method that uses information from a set of training samples in order to design a model employs learning. That's because almost every useful and interesting pattern recognition problem is complex enough, so we can't define a classification decision rule or region automatically. Learning refers to an algorithm whose goal is to reduce the error on a set of training data and can be divided into 3 categories, since pattern recognition problems are not only classification problems, like the ones we described in previous sections. (Duda et al. (2001))

2.2.1 Supervised learning

Applications in which the training data consists of the input feature vectors along with their corresponding target (ground truth) vectors are known as supervised learning problems. If the target (or label) is one of a specific set of classes, then the problem is a classification problem, like the one we described in Section 1.3. For example, if the training samples are handwritten digits (from 0 to 9) and our goal is to recognize them, the target of each training sample is a number from 0 to 9. So, the goal of the classifier is to predict the correct number displayed in the input image. If the target is one or more continuous variables, the problem is a regression problem. For example, if we have a function $y = f(x)$ and the input features are some x_i values, the model is trained to be able to predict the corresponding $y_i = f(x_i)$ values. (Bishop (2006))

2.2.2 Unsupervised learning

Another category of pattern recognition problems includes cases where the input feature vectors don't have a corresponding target vector. These are *unsupervised learning* problems. In this case, the most common problem is to find underlying similarities between the feature vectors and group them in clusters based on these similarities. This process is called clustering. Other unsupervised training problems are to determine the distribution of data in the input space (density estimation) or to project the data from high-dimensional space to lower dimensions, for example for visualization purposes. An interesting issue that arises

in clustering problems is to define and measure the similarity between two feature vectors, as well as to choose the clustering method that will actually group the samples based on the measured similarity.(Theodoridis and Koutroumbas (2009), Bishop (2006))

2.2.3 Semi-supervised learning

Semi-supervised pattern recognition problems have the same goal as supervised learning problems, as far as the classification of patterns is concerned, though the difference is that in semi-supervised learning only a part of the training samples have known targets and the rest don't have corresponding targets. The first ones are called labeled, and the second unlabeled training samples. Semi-supervised pattern recognition has an application in cases where we have access to a limited number of labeled data. This can be the case when we deal with big data sets that need manual annotation of each sample. In case of classification problems, the unlabeled data can be useful in the system design, because they can offer additional information regarding the underlying structure of the training data. In case of clustering problems, labeled data is used as constraints between labeled samples, in the form of must-links and cannot-links. Samples that are connected with a must-link constraint (samples with the same label) must belong to the same cluster, whereas samples that are connected with a cannot-link constraint (samples with different labels) can't belong to the same cluster. (Theodoridis and Koutroumbas (2009))

2.2.4 Reinforcement learning

Reinforcement learning is a machine learning technique, whose goal is to solve the problem of choosing the most suitable action to take in a given environment, in order to maximize a specific reward. In contrast to traditional supervised learning, reinforcement learning doesn't get desired labels of the training inputs, but estimates the best outputs (actions), based on the reward it yields. Often, the current action not only affects the immediate reward, but the long-term reward as well. Reinforcement learning is mostly used to train computers to play games, and they can achieve really high scores compared to human performance. An important issue of reinforcement learning is the trade-off between exploration, in which the system experiments with new kinds of actions to check how profitable or harmful they are, and exploitation, in which the system follows a more conservative approach of selecting already known actions to achieve a high reward. The most efficient approach is to maintain a balance between the two strategies, because focusing on only one of those will result in poor outcomes. Reinforcement learning remains an active area of machine learning research. (Bishop (2006))

2.3 Learning process

Machine learning algorithms work on the basis of trying to learn from experience and training examples by *optimizing a cost function*. This is also called the loss function and is task-dependent. For example, the most straightforward loss function in a classification task is the classification error, which the classifier is trained to minimize. So, the loss function lets us quantify the quality of any particular set of model parameters. An important issue regarding loss functions is how to incorporate knowledge about the cost and the specific task, and the question that subsequently arises is how the choice of loss function will affect the design of the model. (Duda et al. (2001))

2.3.1 Loss functions

We will now go through some basic loss functions, that are widely used in machine learning. Loss functions map an input or some variables to a real number, which represents the cost to be minimized.

2.3.1.1 Regression Loss functions

We have already described the concept of regression problems. The simplest example is to train a model in order to fit a curve $f(x)$ on given points. So, the data set consists of pairs of points of the form (x_i, y_i), $i = 1, ..., N$ and the model must learn to map an input point x_i to its corresponding value y_i through an appropriate function $f(x)$. So, for each point x_i, the real value is y_i and the predicted value from the model is $f(x_i)$.

- Squared Error Loss: Squared Error Loss is the squared difference between the actual and the predicted values
$$L = (y - f(x))^2 \tag{2.17}$$
- Absolute Error Loss: Absolute Error Loss is the absolute value of the difference between the actual and the predicted value:
$$L = |y - f(x)| \tag{2.18}$$

2.3.1.2 Classification Loss functions

In classification problems, every input feature vector has a corresponding label. So the model takes as input the training sample and predicts its label. The basic loss function in this kind of problems is Cross Entropy Loss. First of all, entropy is a quantity that gives information about the uncertainty involved with certain probability distributions; the more uncertainty/variation in a probability distribution, the larger is the entropy. Entropy of a (discrete) distribution p is defined as follows:

$$H(p) = -\sum_x p(x) log(p(x)) \tag{2.19}$$

The negative sign guarantees that the quantity is positive, since $p(x) \leq 1 \Rightarrow log(p(x)) < 0$. Cross Entropy between two distributions p and q is used to quantify their difference. In terms of information theory, it can be seen as the distance between the two distributions, from the aspect of the amount of information (bits) that is needed to explain that distance. It is defined as follows:

$$H(p, q) = -E_p[log(q)] \tag{2.20}$$

For discrete distributions, this relation is equal to:

$$H(p, q) = -\sum_x p(x) log(q(x)) \tag{2.21}$$

In machine learning, we assume the true probability of each pattern i is p_i, and q_i is the predicted value of the model for this particular pattern. For example, if we have to deal with a classification problem, where each pattern belongs to one of two classes, with labels 0 and 1 (binary classification problem), the output of the model can be interpreted as a probability that the current pattern, with input vector x, belongs to class 1. This probability is modeled through the logistic function

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2.22}$$

where z is the actual output of the model, i.e. a function of the input vector x ($z = f(x)$). Thus, the probability that the label of x is 1 is

$$q_{y=1} = \hat{y} = \frac{1}{1 + e^{-f(x)}} \tag{2.23}$$

and as a result, the probability that its label is 0 is

$$q_{y=0} = 1 - \hat{y} \tag{2.24}$$

This means that $p \in \{y, 1-y\}$ (where y is the actual label, $y \in \{0, 1\}$) and $q \in \{\hat{y}, 1-\hat{y}\}$. The cross entropy of these two distributions can now be used to estimate their difference.

$$H(p, q) = -\sum_i p_i log(q_i) = -y \cdot log(\hat{y}) - (1 - y) \cdot log(1 - \hat{y}) \tag{2.25}$$

The greater the difference between y and $\hat{y}$, the greater the value of the cross entropy loss function. For example, if $y = 1$ and $\hat{y} \approx 0$, then the first term of equation 2.25 will have a large value. On the other hand, if $\hat{y} \approx 1$, the first term is almost equal to zero.

This cross entropy loss function we described is used in binary classification problems. However, it can be generalized for multi-class classifications problems as follows:

$$H(p, q) = -\sum_i p_i log(q_i) = -\sum_i y_i log(\hat{y}_i) \tag{2.26}$$

Of course, it is not necessary that y takes only discrete values. If the case is not a classification task, y can take any value between 0 and 1, just like $\hat{y}$. That's why cross entropy loss is also used to measure the error of reconstruction in for example an autoencoder (see Section 2.5.4.2).

Another useful loss function that measures the difference between two distributions p and q is the so-called Kullback-Leibler divergence:

$$KL(p\|q) = \sum_x p(x) \log\left(\frac{p(x)}{q(x)}\right) \tag{2.27}$$

This relation is quite obvious; if p and q are similar, their fraction is close to 1 and thus KL divergence is close to zero indicating p and q have small differences. On the other hand, if they are quite dissimilar, KL divergence will get larger values.

It is interesting to notice that, in fact:

$$KL(p\|q) = H(p, q) - H(p) \tag{2.28}$$

and, since p is the target distribution and as a result, $H(p)$ is the same regardless of distribution q, we can ommit this term and just calculate $H(p, q)$ (cross entropy loss). That's why it is more common to use cross entropy loss than KL divergence.

2.3.2 Optimization

Optimization is the process of choosing the most suitable model parameters in order to minimize the loss function, which is scalar. This problem of minimizing a loss function $L(\mathbf{a})$ with respect to a parameter vector $\mathbf{a}$ can be solved by a *gradient descent procedure*. The idea behind this iterative method is to use gradient information from the loss function, in order to update the parameter vector by comprising a small step in the direction of negative gradient.

So, we start with an arbitrary initial parameter vector $\mathbf{a}(1)$ and compute the gradient vector $\nabla\mathbf{L}(\mathbf{a}(1))$. The next value $\mathbf{a}(2)$ is obtained by moving some distance from $\mathbf{a}(1)$ in the direction of steepest descent. In general, $\mathbf{a}(k+1)$ is calculated from $\mathbf{a}(k)$ by the equation:

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\nabla\mathbf{L}(\mathbf{a}(k)) \tag{2.29}$$

where η is a positive scale factor, called *learning rate*, which sets the step size, and as a consequence how "steep" the actual update is. We want this sequence of parameter vectors to finally converge to a solution minimizing $\mathbf{L}(\mathbf{a})$. Thus, the number of updates (iterations) needed are problem-dependent. A stop criterion often used is that the update quantity $\eta(k)\nabla\mathbf{L}(\mathbf{a}(1))$ is smaller than a threshold θ, chosen by the user.

One of the most important issues regarding gradient descent procedures is the selection of the learning rate $\eta(k)$. In fact, if $\eta(k)$ is too small, the convergence will be too slow, whereas if it is too large the update can overshoot and even diverge.

Another interesting issue about gradient descent methods in machine learning, is that we choose the loss function L based on the training set, so each step requires the use of the whole data set in order to calculate $\nabla\mathbf{L}$. Techniques that use the whole data set at each iteration are called *batch* methods. There is also an online version of gradient descent, which is very useful for large data sets. This method, also called *stochastic* gradient descent updates the parameter vector based on one data point at a time, for example by examining each point sequentially. The most common, though, is to use a batch of data points to calculate $\nabla\mathbf{L}$, which combines the advantages of the two aforementioned methods: solves the problem of large data sets, while incorporating parallelism thanks to using groups of samples at each step, which speeds up the training process. (Duda et al. (2001), Bishop (2006))

Gradient descent optimization algorithms are numerous (Adaline, Adamax, Adam, RMSprop etc), each one inserting a small variation to the classical approach. One of the most widely used is Adam (Adaptive Moment Estimation), which updates the weights using estimations of the first two moments of past gradients (mean and standard deviation).

2.4 Classifiers

As already mentioned, classifiers are used in supervised learning applications and their goal is to classify a sample $\boldsymbol{x}$ to a category y. This means that, for each sample $\boldsymbol{x}_i$, $i = 1, ..., N$ their goal is to find the class y_k, $k = 1, ..., M$ that maximizes the probability $p(y_k|\boldsymbol{x}_i)$ (*decision theory*):

$$\hat{y} = \arg\max_{y_k} p(y_k|\boldsymbol{x}_i) \tag{2.30}$$

There are two kinds of classifiers trying to calculate this probability, the discriminative and the generative models.

- Discriminative models determine the posterior class probabilities $p(y_k|\boldsymbol{x}_i)$ directly and then apply decision theory (Equation 2.30) to assign each $\boldsymbol{x}_i$ to the most probable class. The most well-known discriminative models are Logistic Regression, Support Vector Machines (SVMs), perceptrons (see Section 2.5.1) and traditional neural networks (see Section 2.5.2).

- Generative models also calculate the posterior class probabilities, but with a different approach. They first determine the class-conditional probability densities $p(\boldsymbol{x}_i|y_k)$ for each class y_k and then the prior probabilities $p(y_k)$. So, using the Bayes rule, the posterior probability is calculated as:

$$p(y_k|\boldsymbol{x}_i) = \frac{p(\boldsymbol{x}_i|y_k)p(y_k)}{p(\boldsymbol{x}_i)} = \frac{p(\boldsymbol{x}_i|y_k)p(y_k)}{\sum_k p(\boldsymbol{x}_i|y_k)p(y_k)} \tag{2.31}$$

Alternatively, they calculate the joint probability $p(\boldsymbol{x}_i, y_k)$ and then obtain the posterior probability by normalizing. The most common generative models are Naive Bayes classifier, Bayesian Networks and Hidden Markov Models (HMMs).

It is obvious that generative models don't just learn a decision boundary, like discriminative models, but also the underlying distribution of each class. The choice of one of the two categories depends on the task and the application, but discriminative models are more common in general. Logistic Regression and SVMs, in particular, are very popular.

Logistic Regression is a linear model which, in a two-class problem, assumes $p(y_1|\boldsymbol{x})$ can be written as a logistic sigmoid acting on a linear function of the feature vector $\boldsymbol{x}$:

$$p(y_1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}) \tag{2.32}$$

This linear function $\boldsymbol{w}^T\boldsymbol{x}$ is the decision boundary (discriminant function) of the classifier, meaning that points that lie on the one side of this hyperplane belong to class y_1 and the others to class y_2. Obviously, $p(y_2|\boldsymbol{x}) = 1 - p(y_1|\boldsymbol{x})$. The parameters of the model are the components of vector $\boldsymbol{w}$ and are computed using the Maximum Likelihood Estimation and the Cross Entropy Loss function (see Section 2.3.1.2).

SVMs are again linear models, trying to determine the weight vector of the linear discriminant function $g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$. The goal of SVMs is to find the weights that maximize the margin between the hyperplane defined by the linear function and the training samples of the two classes, which is equal to $\frac{\left\|g(\boldsymbol{x})\right\|}{\left\|\boldsymbol{w}\right\|}$. In general, the larger the margin, the better the generalization capability of the classifier. SVMs can also overcome the problem of non-linearly separable data by transforming the samples to a space where they are linearly separable, by applying a transform ϕ. In this case, $\boldsymbol{z} = \phi(\boldsymbol{x})$ is the transformed vector, and the linear discriminant function is $g(\boldsymbol{z}) = \boldsymbol{w}^T\boldsymbol{z}$. (Bishop (2006), Duda et al. (2001))

2.5 Deep learning

As discussed above, every model or learning technique requires some input feature vectors. The feature extraction procedure is problem-dependent, and therefore it is often tricky to discover the most appropriate representations that will simplify the solution to the learning problem, regardless if it is supervised or unsupervised. One idea is to employ machine learning to not only train the computer to map the feature vector to a desired output, but also to learn the representation itself. Towards this end, deep learning can give the solution to extracting high-level features from raw data by introducing representations that are expressed in terms of other, simpler representations. So deep learning enables the computer to build complex concepts out of simple ones. Deep learning is actually a subfield of machine learning inspired by the structure and function of the human brain and the way humans think. (Goodfellow et al. (2016))

The essence of deep learning is the use of a multitude of elemental non-linear computing elements, known as *artificial neurons*, organized as networks, such as the structure of their interconnections resemble the way neurons are interconnected in human brains. These are the so-called *neural networks*, which are trained via successive presentations of training patterns.

The interest in neural networks started way back, with the development of learning machines called *perceptrons* during the 1950s and 1960s, that imitated the way brain neurons work. More specifically, mathematical proofs were found showing that perceptrons, can converge to a solution after a finite number of steps, when trained with linearly separable data. The solution took the form of parameters (coefficients) of hyperplanes that were suitable for separating the data into the classes describing the training samples.

Unfortunately, perceptrons could not guarantee successful results in case of non-linearly separable data. That's why the idea was to employ multilayers of perceptrons, since they could possibly learn to separate data with more complex-related representations. True revolution lies in the fact that these networks could now learn the representations which are more suitable for recognition of the input data. Each layer of the network actually refines the representations to more abstract levels. This multilayer training is referred to as *deep learning*, and its practical implementations are mostly associated with large data sets.

The effective training method that enables the learning of the representations is called *backpropagation*. It hasn't been proven that this algorithm converges to a solution with the mathematical rigor achieved in single layer perceptrons, but it has produced remarkable results for the field of pattern recognition.

Of course, although neural networks might be highly autonomous at their training, they actually require parameter tuning done by humans. Configurable parameters are the number of layers, the number of neurons per layer and other problem dependent coefficients.

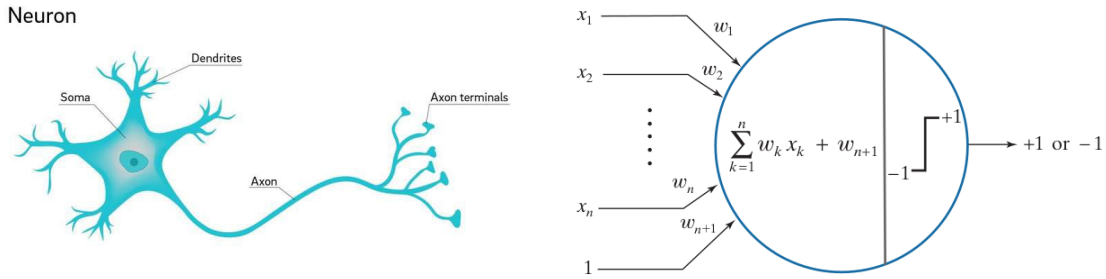
Deep learning doesn't always provide the best possible solution; there are numerous applications that are better handled by more traditional methods. However, it has been proven extremely useful in applications that have been challenging for other methods. In fact, it has offered the opportunity to solve many problems in various domains and has been used in fields like speech recognition, natural language processing and understanding, genetics etc. (Gonzalez and Woods (2008))

We are now going to get an insight into the way perceptrons work and are then combined to create neural networks.

2.5.1 The Perceptron

2.5.1.1 Biological neurons vs perceptrons

The perceptron is a mathematical model of a biological neuron.



(a) A biological neuron.

(b) A perceptron.

Figure 2.5: A biological neuron compared to a perceptron.

- In actual neurons, the dendrite receives electrical signals from axons of other neurons. These signals are modeled as numerical values in the perceptron case. These values x_i, $i = 1, 2, ..., n$ are interpreted as elements of an n-dimensional input vector $\mathbf{x}$.
- At the synapses between the dendrite and the axons of other neurons, electrical signals are modulated in various amounts. This is modeled in perceptron units by multiplying each input value x_i by a *weight* w_i.
- An actual neuron gives an output signal only if the total strength of the input exceeds a certain threshold. The total strength in perceptron is modeled as the weighted sum of the input values. The weighted sum can be expressed in three different ways:

$$w_1x_1 + w_2x_2 + ... + w_nx_n + w_{n+1} = \sum_{i=1}^{n} w_ix_i + w_{n+1} = \mathbf{w}^T\mathbf{x} + w_{n+1} \tag{2.33}$$

We then apply a step function on the sum to determine whether the output will be equal to 1 (activated) or to -1 (not activated). The output f of the neuron is described by the following equation:

$$f(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{w}^T\mathbf{x} + w_{n+1} > 0 \\ -1, & \text{if } \mathbf{w}^T\mathbf{x} + w_{n+1} < 0 \end{cases} \tag{2.34}$$

2.5.1.2 Perceptron used in linear classification problems

It is obvious that the weighted sum calculated in the perceptron corresponds to a linear boundary (hyperplane) in n-dimensional space:

$$\mathbf{w}^T\mathbf{x} + w_{n+1} = 0 \tag{2.35}$$

where $\mathbf{w}$ (also referred to as weight vector, whereas w_{n+1} as bias) and $\mathbf{x}$ are n-dimensional column vectors and $\mathbf{w}^T\mathbf{x}$ is their inner product. This means that we can employ a single perceptron unit to solve a classification problem by learning this linear boundary between linearly separable pattern classes.

If we add a 1 at the end of every pattern vector, then $\mathbf{x} = [x_1, x_2, ..., x_n, 1]^T$ and $\mathbf{w} = [w_1, w_2, ..., w_n, w_{n+1}]^T$. So, the classification problem between two linearly separable pattern classes c_1 and c_2 is to find a set of weights $\mathbf{w}$ that, given an input vector $\mathbf{x}$ satisfy the following property:

$$\mathbf{w}^T\mathbf{x} = \begin{cases} > 0, & \text{if } \mathbf{x} \in c_1 \\ < 0, & \text{if } \mathbf{x} \in c_2 \end{cases} \tag{2.36}$$

In this formulation, $\mathbf{x}$ and $\mathbf{w}$ are referred to as *augmented* pattern and weight vectors, respectively. The solution to the aforementioned problem is given through an iterative algorithm, which according to the *perceptron convergence theorem* will surely converge to a solution (a set of weights that define a hyperplane) after a finite number of steps, if the pattern classes are linearly separable.

The so-called *perceptron training algorithm* is quite simple. We let α denote the *learning rate*, which is a parameter defining how steep the weight vector updates will be in each iteration. The initial values of the weight vector, denoted by $\mathbf{w}(1)$, are arbitrary. Let's say our data set consists of N patterns $\mathbf{x}_j$, $j = 1, 2, ..., N$. We do the following for $k = 2, 3, ...$:

For each pattern vector $\mathbf{x}_j$, at step k:

1. If $\mathbf{x}_j \in c_1$ and $\mathbf{w}^T\mathbf{x}_j \leq 0$, let:

$$\mathbf{w}(k+1) = \mathbf{w} + \alpha\mathbf{x}_j \tag{2.37}$$

2. If $\mathbf{x}_j \in c_2$ and $\mathbf{w}^T\mathbf{x}_j \geq 0$, let:

$$\mathbf{w}(k+1) = \mathbf{w} - \alpha\mathbf{x}_j \tag{2.38}$$

3. Otherwise, let

$$\mathbf{w}(k+1) = \mathbf{w}(k) \tag{2.39}$$

The concept behind this algorithm is that if a pattern is misclassified, we are trying to shift the weight vector to a direction that increases the probability of correct classification the next time the specific pattern is presented. That's also why if the classification of a pattern is correct, no change is applied to the weight vector. The algorithm converges and terminates at step K when all patterns of our data set can be correctly classified by using the current weight vector $\mathbf{w}(K)$. (Gonzalez and Woods (2008))

2.5.2 Multilayer Feedforward Neural Networks

In practice, linearly separable data are a rare occasion, which means that the simple perceptron unit we went through in previous Section can't provide a solution. The natural question that comes to mind after this realization is if some sort of combination of multiple perceptrons can learn decision boundaries (functions) that successfully classify non-linearly separable data. It has been found that this is actually the case with multilayer feedforward neural networks.

2.5.2.1 Artificial Neuron

If we combine (connect) perceptron-like units, we can form *neural networks*. These units are called *artificial neurons* and function in the same way as perceptrons, though they differ in the way they process the result of the computations. More specifically, instead of using a hard thresholding function (step function), which outputs only two values (1 and -1), they employ a more soft function, as explained below. For example, let z denote the output of the perceptron before thresholding, i.e. $z = \sum_{i=1}^{n} w_i x_i + w_{n+1}$ and h the function employed to calculate the neuron's final output. We call h the activation function of the unit. The total output is denoted by a and is the result of passing z to h ($a = h(z)$).

If the output of a perceptron before thresholding (z_k) has a value a little greater than zero (then $a = h(z_k) = +1$), and another output z_l has a value a little smaller than zero (then $a = h(z_l) = -1$), they will result in a big swing in the perceptron's output, although z_k and z_l were quite close. This instability which can occur with perceptrons should be avoided, since neural networks consist of layers of neurons, in which the output of one affects the behavior and eventual outputs of all the following neurons.

We can overcome this problem by changing the step activation function to a smoother function, for example the sigmoid function, described by equation:

$$h(z) = \frac{1}{1 + e^{-z}} \tag{2.40}$$

where $z = \sum_{i=1}^{n} w_i x_i + w_{n+1}$. It is obvious that the only difference between perceptron and the artificial neuron is the activation function they use for processing z. Now, w_{n+1} is denoted by b (*bias*). (Gonzalez and Woods (2008)) Figure 2.6 shows an artificial neuron with a sigmoid activation function.

2.5.2.2 Activation functions

The sigmoid function is used very frequently as the activation function of artificial neurons. There are also other functions with the required properties that are often used. For example, the hyperbolic tangent has the same shape as the sigmoid function, but is symmetric about both axes (see Figure 2.7).

$$h(z) = \tanh(z) \tag{2.41}$$

The Rectified Linear Unit (ReLU) activation function has also become very popular. ReLU keeps all the positive inputs unchanged and sets all negative inputs to zero:

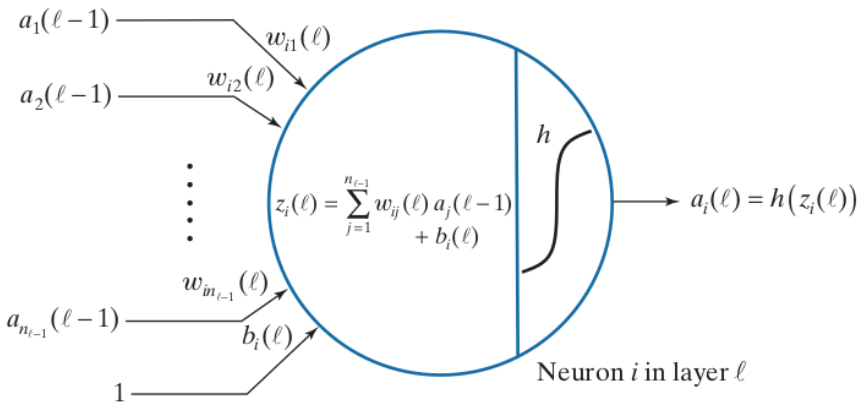


Figure 2.6: An artificial neuron, whose inputs are the outputs of preceding neurons and whose activation function is the sigmoid function. (Gonzalez and Woods (2008))

$$h(z) = \max(0, z) \tag{2.42}$$

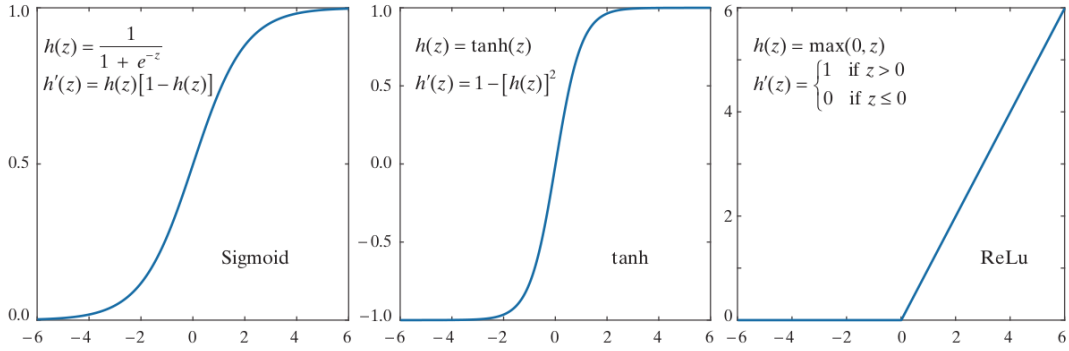


Figure 2.7: Sigmoid, Hyperbolic tangent and ReLU activation functions. Their derivatives are shown since they are useful for the backpropagation algorithm discussed in Section 2.5.2.3. (Gonzalez and Woods (2008))

2.5.2.3 Fully Connected Neural Network

As stated above, artificial neurons are interconnected to form a multilayer feedforward neural network. The architecture of such a network is presented in Figure 2.8.

The network consists of multiple *layers*. For example, the network shown in Figure 2.8 consists of L layers, where each layer consists of a custom number of neurons, except for the first layer, where nodes are just the inputs $x_1, x_2, ..., x_n$ of the network. Since the output of the neurons in layers 2 to $L-1$ aren't known, these are called *hidden* layers. The essential characteristic of the network is that information flows from the left to the right (that's why it's called feedforward) and that each neuron is connected to all preceding and following neurons, i.e. all neurons of the previous and the next layer, and only them. That's why it's called fully connected.

Forward pass through a Feedforward Neural Network A forward pass through a Feedforward Neural Network maps the input layer (the values of vector $\mathbf{x}$) to the output

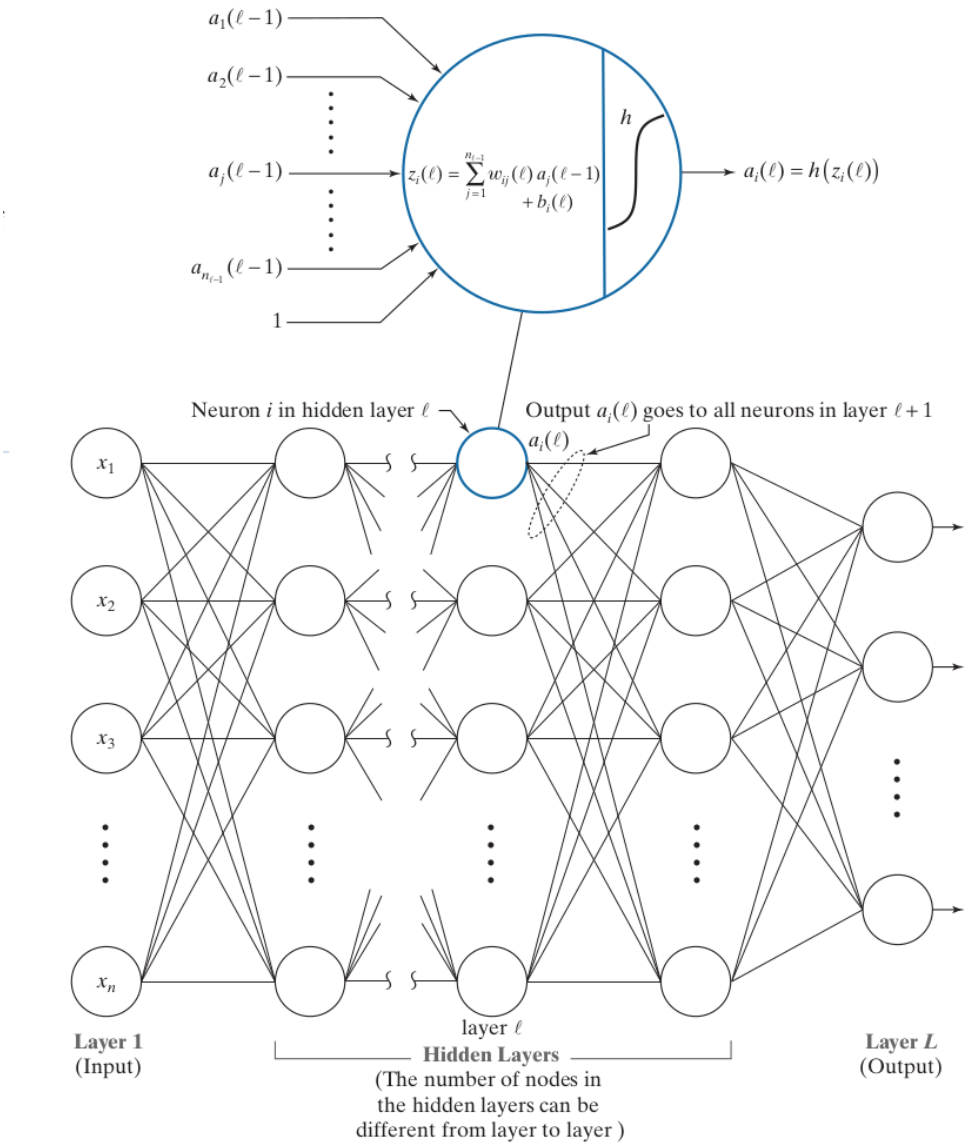


Figure 2.8: Fully Connected Neural Network. (Gonzalez and Woods (2008))

layer.

We use l as the index that describes the layer we are focusing on. Each layer l consists of n_l neurons, where the output of neuron j, $j = 1, 2, ..., n_l$ is denoted by a_j . If $l = 1$, the output of the input layer is the components of the vector $\mathbf{x}$:

$$a_j(1) = x_j, \quad j = 1, 2, ..., n_1 \tag{2.43}$$

For layers $l > 1$, each neuron's i, $i = 1, 2, ..., n_l$ inputs are the outputs of the neurons of the previous layer ($a_j(l-1)$, $j = 1, 2, ..., n_{l-1}$). These inputs are then multiplied by a weight w_{ij} (i is the neuron that receives the signal and j the neuron that sends it). Also, each neuron i has a bias value b_i that is added to the weighted sum. Let $z_i(l)$ denote the output of neuron i in layer l before applying the activation function.

$$z_i(l) = \sum_{j=1}^{n_{l-1}} w_{ij}(l) a_j(l-1) + b_i(l) \tag{2.44}$$

for $i = 1, 2, ..., n_l$ and $l = 2, ..., L$. The final output of the neuron is the *activation value*:

$$a_i(l) = h(z_i(l)) \tag{2.45}$$

The output of the whole network is the output of the nodes of the final layer L.

$$a_i(L) = h(z_i(L)) \tag{2.46}$$

for $i = 1, ..., n_L$.

The Backpropagation method A Feedforward Neural Network is fully described by its weights, biases and activation function; thus, training a neural network refers to employing training examples to learn these parameters. This training procedure is achieved with the *backpropagation* method, which consists of 4 steps:

1. Inputting the training pattern vectors.
2. A forward pass through the network to classify the patterns and calculate the classification error.
3. A backward (backpropagation) pass that feeds the output error back through the network in order to compute the necessary quantities to update the parameters.
4. Updating the weights and biases of the network.

The aforementioned steps are repeated until the error is below a specific, accepted threshold.

Goal of the training process, as is already mentioned in previous sections, is to minimize an error function, depending on the nature of the problem we are trying to solve (see Section 2.3.1). In case of classification problems, Cross Entropy Loss function is the most common to use. The procedure of calculating the probability that a specific pattern belongs to a certain class, described in equations 2.22 - 2.24 is achieved by using the sigmoid activation function in the last layer of the network.

If $\mathbf{r}$ denotes the desired response ($\mathbf{r} = [r_1(L), r_2(L), ..., r_{n_L}(L)]^T$) and $\mathbf{a}(L)$ the vector consisting of the outputs of the neurons of the final layer L ($\mathbf{a} = [a_1(L), a_2(L), ..., a_{n_L}(L)]^T$), the total error function is defined as:

$$E = -\sum_{i=1}^{n_L} r_i(L)\log(a_i(L)) \tag{2.47}$$

In case of a regression problem , the total error function is defined as:

$$E = \frac{1}{2}\sum_{j=1}^{j=n_L} (r_j - a_j(L))^2 = \frac{1}{2}\left\|\mathbf{r} - \mathbf{a}(L)\right\|^2 \tag{2.48}$$

In order to find the weights and biases that minimize this error, we use the optimization method gradient descent, which we discussed in Section 2.3.2. According to gradient descent, the update equations for the weight w_{ij} and bias b_i in layer l are:

$$w_{ij}(l) = w_{ij}(l) - \alpha\frac{\partial E}{\partial w_{ij}(l)} \tag{2.49}$$

$$b_i(l) = b_i(l) - \alpha\frac{\partial E}{\partial b_i(l)} \tag{2.50}$$

where α is the learning rate.

However, it is not straightforward to get the gradients of E with respect to the weights and biases in hidden layers. That's why we need to propagate the output error back to the network. Since E isn't an immediate function of $w_{ij}(l)$, we use the *chain rule* to calculate the partial derivative:

$$\frac{\partial E}{\partial w_{ij}(l)} = \frac{\partial E}{\partial z_i(l)} \frac{\partial z_i(l)}{\partial w_{ij}(l)} = \frac{\partial E}{\partial z_i(l)} a_j(l-1) \tag{2.51}$$

To calculate $\frac{\partial E}{\partial z_i(l)}$ we use the chain rule again. Since the node i of layer l is connected to all nodes of layer $l+1$:

$$\frac{\partial E}{\partial z_i(l)} = \sum_j \frac{\partial E}{\partial z_j(l+1)} \frac{\partial z_j(l+1)}{\partial a_i(l)} \frac{\partial a_i(l)}{\partial z_i(l)} = \sum_j \frac{\partial E}{\partial z_j(l+1)} w_{ij} \frac{\partial h(z_i(l))}{\partial z_i(l)} \tag{2.52}$$

The equation 2.52 suggests that the calculation of $\frac{\partial E}{\partial z_i(l)}$ is actually recursive, using the partial derivatives of E with respect to all z_j, $j = 1, ..., n_{l+1}$ of the next layer $l+1$. This is calculated by starting from the final layer L and going backwards, where:

$$\frac{\partial E}{\partial z_i(L)} = \frac{\partial E}{\partial a_i(L)} \frac{\partial a_i(L)}{\partial z_i(L)} = \frac{\partial E}{\partial a_i(L)} \frac{\partial h(z_i(L))}{\partial z_i(L)} \tag{2.53}$$

So, the equations above describe the way the error is backpropagated to the network, and how the update of each weight and bias is calculated, resulting in the network learning them in order to classify the training patterns as successfully as possible, even if they are *not* linearly separable. (Gonzalez and Woods (2008))

2.5.3 Deep Convolutional Neural Networks

Up to this point, pattern features are vectors, that neural networks take as input. For example, in the case of image classification, it means that features must have been extracted from images before they are used as input of a neural network. However, the true value of neural networks lies in the fact that they can learn the features directly from training data, on their own. In this section we are going to focus on a class of neural networks, called *Deep Convolutional Neural Networks* (CNNs), that accept raw images as input, learn the features of the images and can then be used for image classification or other applications.

Convolutional Neural Networks were inspired by biological processes, since the connectivity of the neurons resemble the organization of animal visual cortex.

2.5.3.1 Fully Connected Neural Networks vs CNNs

Fully Connected Neural Networks and CNNs share both some similarities and some differences.

Their basic similarity is that the computations performed in both networks are similar: a sum of products plus a bias is calculated, it passes through an activation function and the activation value is a single input to the next layer.

The main differences are the following:

- CNNs accept as inputs 2d arrays (images), while inputs to Fully Connected Neural Networks are vectors.
- CNNs can learn the important features from raw images and don't need to get pre-extracted feature vectors, which is a great advantage, since they don't require prior knowledge on the data set of images. CNNs focus on learning the features important for our *specific task*.

- The layers are connected differently. As already mentioned, in Fully Connected Neural Networks, the output of a neuron is fed to all neurons of the following layer. In CNNs, a convolution over the spatial neighborhood in the output of a layer calculates a single value, which is fed to the following layer. That's why CNNs are not fully connected, in the sense described above.

- The 2-D arrays that pass from one layer to another in CNNs are subsampled to reduce sensitivity to spatial translations of the input.

2.5.3.2 The spatial convolution

The basis of CNN operation is spatial convolution. Spatial convolution calculates the sum of products between image pixels and a kernel consisting of weights. More specifically, the convolution of a kernel w of size mxn with an image $g(x, y)$ is written as $(w * g)(x, y)$ and described by the following equation:

$$(w * g)(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) g(x - s, y - t) \tag{2.54}$$

where $a = \frac{n-1}{2}$ and $b = \frac{m-1}{2}$. In simpler words, we have a position (x, y) of the image, we overlap the kernel on the image so that its center is on (x, y) and every weight corresponds to an image pixel and then we calculate the sum of the products of each weight of the kernel with the corresponding image pixel.

The result is a scalar value calculated at every spatial location (x, y) of the input image. If we add a bias and pass the result through an activation function, the similarities between this procedure and the way neurons in Feedforward Neural Networks perform their computations are obvious.

2.5.3.3 Forward pass through a CNN

We consider a CNN consisting of L layers, where operations of each layer are explained below. As mentioned before, the input of a CNN is a raw image, for example of WxHxD_0 dimensions. The depth D_0 of the image refers to its different components (e.g. RGB images consist of 3 components (2-D arrays)).

Convolutional layer The first part of each CNN layer is the convolutional layer, where the convolution operation is performed. The convolution operation takes place at every location (x, y) of the image, taking into account a neighborhood of this location. This neighborhood is called *receptive field* and selects a region of pixels of the image, around (x, y). A set of weights, arranged in the same shape as the receptive field and known as a *kernel*, slides along with the receptive field over the image and, for each location (x, y), the image region defined by the receptive field and the kernel are convolved.

Hyperparameters of this layer are the following:

- Number of kernels (also referred to as filters) (D_l, where l refers to the current layer). The number of kernels define the depth of the output of the convolutional layer.

- Filter size (FxFxD_l). The filter size is the same as the size of the receptive field and the depth is the same as the depth of the input. It is common that the filter has the same width and height. In general, the filter size declares how many neighbors of each location we use in order to calculate the new value of the location. So, the larger the kernel, the more information of the input layer each number in the output layer carries.

- Stride (S). The stride is the number of spatial incremental steps we take when we slide the receptive field (and the kernel) horizontally and vertically over the input.
- Padding (P). Convolution of the input with the kernel results in an image with reduced size, because the kernel has to fit inside the input array, meaning that the kernel can't be centered on pixels of the first or last row or column, for example. In order to preserve the size of the output of the convolutional layer equal to the size of the input, we expand the input array by adding a suitable number (P) of rows and columns filled with zeros, on its perimeter, before the convolution operation is performed.

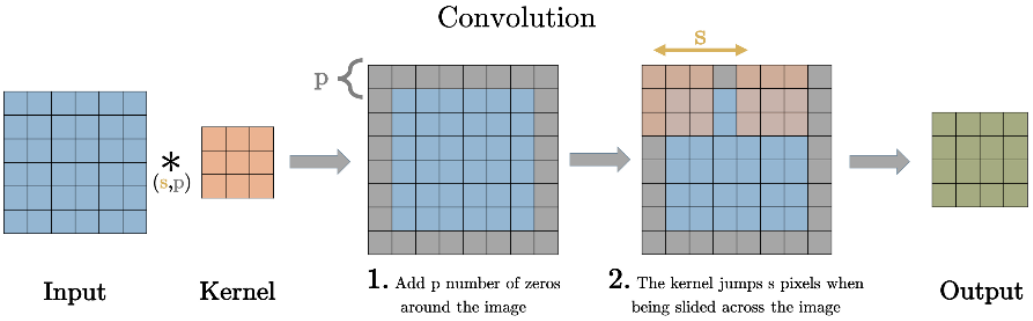


Figure 2.9: The convolution operation.

If $a_{x,y,d}(l-1)$ denotes the input value of layer l at position (x, y) and depth d, $d = 1, ..., D_{l-1}$, and $w_i(l)$, $i = 1, ..., D_l$, denotes the weights of the kernel i, arranged in the shape of the receptive field, then the convolution result of kernel i with the input image is calculated as:

$$\sum_{d=1}^{D_{l-1}} w_{d,i}(l) * a_{x,y,d}(l-1) = \sum_{d=1}^{D_{l-1}} \sum_{l=-\frac{N-1}{2}}^{\frac{N-1}{2}} \sum_{k=-\frac{N-1}{2}}^{\frac{N-1}{2}} w_{l,k,d,i}(l) a_{x-l,y-k,d}(l) \tag{2.55}$$

where k and l span the dimensions of the kernel. If we add a bias $b_{d,i}$ to the result above, we get $z_{x,y,i}$:

$$z_{x,y,i}(l) = \sum_{d=1}^{D} [w_{d,i}(l) * a_{x,y,d}(l-1) + b_{d,i}] \tag{2.56}$$

In this way, $z_{x,y,i}$ is quite similar to z_i we calculated in Feedforward Neural Networks, since it is simply a weighted sum plus a bias. Then, this result is fed to an activation function:

$$a_{x,y,i}(l) = h(z_{x,y,i}(l)) \tag{2.57}$$

These calculations are done for each of the D_l kernels of layer l, resulting in D_l different outputs, which are then stacked (D_l is the depth of the output of the convolutional layer l). However, W_l and H_l of the output depend on the kernel size, the stride and the pooling:

$$W_l = \left\lfloor \frac{W_{l-1} + 2P_l - F_l}{S_l} + 1 \right\rfloor \tag{2.58}$$

$$H_l = \left\lfloor \frac{H_{l-1} + 2P_l - F_l}{S_l} + 1 \right\rfloor \tag{2.59}$$

Another significant observation is that $a_{x,y,d}(l)$ is calculated using information only from a neighborhood of (x, y) (receptive field), which points out the local connectivity and spatial locality used and emphasized in CNNs, compared to the full connectivity in Feedforward Neural Networks.

This procedure (convolution and feeding the output to the next layer) is quite similar to the way a neuron in visual cortex responds to stimulus. After it is done for all x, y ($x = 1, ..., W_l$, $y = 1, ..., H_l$), a new W_lxH_lxD_l array $a(l)$ has been filled, which consists of $a_{x,y,d}(l)$. This is called a *feature map*, because through convolution some features of the image are actually learned. The weights of the kernel and the bias are the same for all receptive fields of the image, since we want this specific kernel to be responsible for the detection of one specific feature at all locations of the image. A different set of weights and a different bias result in constructing a new feature map, which will contain a new set of features of the initial image. We want the network to learn the weights and biases, so that the automated feature extraction procedure is optimized. This means that finally, the output of convolutional layer l consists of D_l feature maps, with size W_lxH_l each.

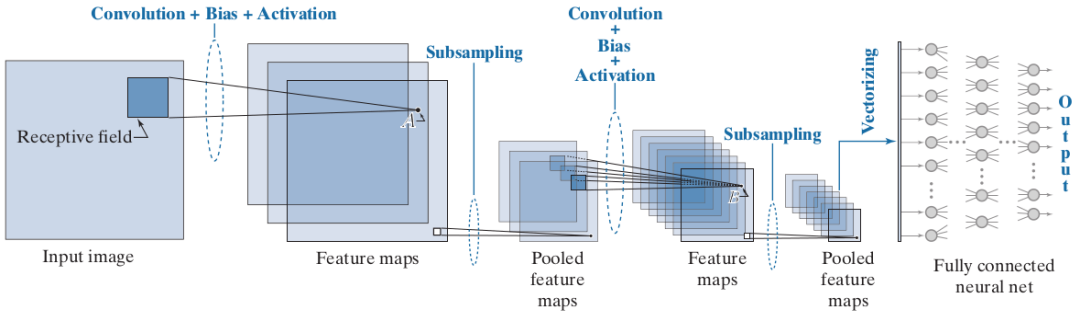


Figure 2.10: A Convolutional Neural Network. (Gonzalez and Woods (2008))

Pooling layer The second operation of a CNN layer is subsampling the resulting feature maps, also referred to as *pooling*. This procedure is useful, because it achieves translational invariance, meaning that if a small translation occurs, most of the pooled values won't change. This property is important, especially if we are interested in whether a specific feature is present in the image, rather than where exactly it is. Pooling is also beneficial, because it gradually reduces the amount of data that needs to be processed, as well as the number of trainable parameters, preventing overfitting. The result of pooling is the pooled feature maps. Pooling is done by dividing each feature map in non-overlapping small regions (typically 2x2) and replace this whole area with one new value, depending on the kind of pooling we choose. There are 3 main kinds of pooling:

- *Max pooling*: the value chosen is the maximum of the elements of the neighborhood
- *Average pooling*: the value chosen is the average of the elements of the neighborhood
- *L2 pooling*: the value chosen is the square root of the sum of the squared neighborhood elements

Hyperparameters used in this layer are:

- Filter size (F). The filter here doesn't have the same meaning as in the convolutional layer, since it just defines the region that will be replaced by a single value and doesn't contain any trainable parameters.

- Stride (S). Again, it refers to the number of steps to use while sliding the filter over the feature map.
- Padding (P). It refers to how many rows and columns of zeros we use for padding the feature map, though padding isn't common in this layer.

It is obvious that if $F = 2$ and $S = 2$, the pooling procedure results in a pooled feature map of dimensions $\frac{W}{2}$x$\frac{H}{2}$. For example, in the case of max pooling, using a 2x2 region with a stride of 2, the pooled feature map derived by feature map $a_d(l)$, $d = 1, ..., D_l$ has values:

$$p_{x,y,d}(l) = \max_{k,l=0,...,1} a_{2x+k,2y+l,d}(l) \tag{2.60}$$

Thus, a CNN layer consists of a convolutional layer and a pooling layer. The equations presented above are the same for every layer l of the network. The resulting pooled feature maps are now the input of the next CNN layer.

The most usual application of CNNs is to use the derived features for classification. That's why it's common that the 2-D output of the last pooling layer is vectorized and then fed to a Fully Connected Feedforward Neural Network, which is responsible for determining the most likely pattern class for the input image.

Transposed Convolutional Layer Transposed convolutional layers are not used in typical CNNs. In our architecture, though, they are necessary, as we will describe in Section 2.5.4.2. Goal of transposed convolutional layers is upsampling, i.e. to generate an output feature map $a(l)$ ($W(l)$x$H(l)$) with greater spatial dimensions than the input feature map $a(l-1)$ ($W(l-1)$x$H(l-1)$). Stride (s) and padding (p) also play an essential role in defining this layer, though they don't have the same usage as in traditional convolutional layers.

More specifically, we introduce 3 new quantities, $z = s - 1$, $p' = k - p - 1$ ($k = F$(size of the kernel is k x k) and $s' = 1$.

At first, we insert z rows and z columns of zeros between every row and column of the input array respectively. The result of this operation is a new feature map $a(l)$ of shape $(2 \cdot W(l-1) - 1)$x$(2 \cdot H(l-1) - 1)$. So, the stride here determines how fast the kernel moves on the output layer in contrast to stride in traditional convolutional layers, where it determines how fast the kernel moves on the input. The kernel size has also a different sense: here, it defines how much we disperse information from input layer to output layer. This means that the larger the kernel, the larger the output. We then pad the resulting image with p' rows (horizontally) and columns (vertically) and calculate the classical convolution with stride $s' = 1$. In this way, as it is easy to observe from Figure 2.11, the input image is upsampled.

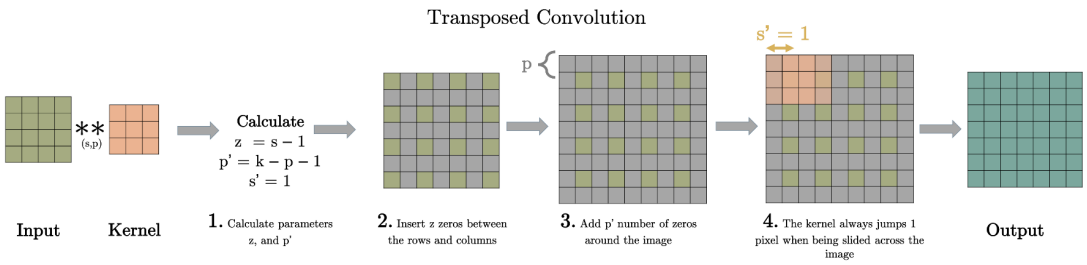


Figure 2.11: The transposed convolution operation.

Transposed convolutional layers are especially useful for image upsampling and also reconstruction in deep architectures, because they involve weights that can be trained to

learn which features of the image are important for an efficient reconstruction. This is more flexible compared to simple interpolation for upsampling.

Backpropagation to train CNNs As already mentioned, CNNs share some similarities with Feedforward Neural Networks. In this sense, a CNN is also trained with the backpropagation method. The error used for training the network is usually the classification error, taking into account the actual pattern class and the output of the feedforward part of the network. The chain rule is also applied here, to get the derivatives of the error with respect to the weights of the kernels and the biases, but since simple multiplication in Feedforward Neural Networks is substituted by convolution in CNNs, the equations are more complex. It is important to note that backpropagation affects only the parameters of the convolutional and not the pooling layer.(Gonzalez and Woods (2008), Goodfellow et al. (2016))

2.5.3.4 Applications

CNNs can be used for every application that requires feature extraction from images. Some of them are presented below:

- Image recognition. CNNs are successfully used for fast and accurate object detection in images. They are also widely used in facial recognition.
- Analyzing documents. CNNs can be used for recognition of handwriting or signatures in documents.

In general, they have replaced many traditional computer vision techniques for feature extraction in specific problems, because their use reduces human intervention in the extraction pipeline.

As mentioned above, it is very common that Convolutional Neural Networks are used for feature extraction for classification purposes and the error used for training the weights is the classification error. However, this is not always the case; in our problem for example, we need to extract features from images, without having a target value. This is achieved by a type of neural networks called *autoencoders*.

2.5.4 Autoencoders

Autoencoders are neural networks that are trained to attempt to copy their input to their output. An autoencoder consists of 3 components: encoder, code and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code. The encoder is described by a function $\mathbf{h} = f(\mathbf{x})$, where $\mathbf{x}$ is the input and $\mathbf{h}$ is the code and the decoder produces a reconstruction $r = g(\mathbf{h})$.

The encoder maps the input to a code that is assumed to contain the most important information (features) of the input and is used to represent it, and then decoder constructs the output from this representation. The code is also referred to as the latent-space representation and can be used as a feature vector to numerous applications.

The goal of the autoencoder is not to just learn the composite function $g(f(\mathbf{x})) = \mathbf{x}$, in the sense that we don't want it to learn to recreate the input perfectly. In fact, we want the autoencoder to learn a useful representation of the input that will contain the most important information from it. (Goodfellow et al. (2016))

2.5.4.1 Undercomplete Autoencoders

One way to obtain useful features from the autoencoder is to constrain $\mathbf{h}$ to have a smaller dimension than $\mathbf{x}$, since in this way the model is forced to prioritize the aspects

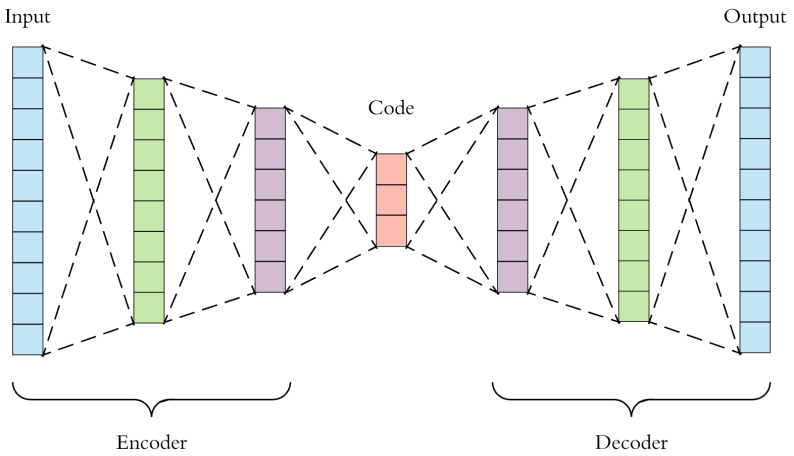


Figure 2.12: The structure of a typical autoencoder.

of the input that they will be copied and as a result to learn significant properties of the input data. An autoencoder whose code dimension is less than the input dimension is called undercomplete. Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data. The learning process is described simply as minimizing a loss function $L(\mathbf{x}, g(f(\mathbf{x})))$, where L is a loss function penalizing $g(f(\mathbf{x}))$ for being dissimilar from $\mathbf{x}$, like the mean squared error function. (Goodfellow et al. (2016))

Commonly, the encoder and the decoder are feed forward neural networks, whose task is to learn the function f and g respectively. In our case, where the inputs are images, the encoder and the decoder are convolutional neural networks, so the whole network is referred to as a *convolutional autoencoder*.

2.5.4.2 Convolutional Autoencoder

A convolutional autoencoder is a type of undercomplete autoencoders, since the intermediate representation has a smaller size than the initial image. The CNN-encoder tries to learn the weights and biases that will result in a code $\mathbf{h}$, which will contain the most useful features for a successful reconstruction of the image by the decoder.

In particular, the CNN-encoder is a normal CNN, with convolutional and pooling layers, as we described in Section 2.5.3. The pooling layers achieve the size reduction of the initial image as we go deeper into the network, which guarantees that the code will indeed have lower dimensions compared to the input image. The code can then be flattened and used as a feature vector which uniquely describes the input image.

The CNN-decoder is again a CNN, but it only consists of transposed convolutional layers. This makes sense, because the goal of the decoder is to learn the appropriate filters that will be able to efficiently reconstruct the image from the code. Of course, pooling layers aren't a part of this type of CNN, since we want to upsample the feature maps.

The convolutional autoencoder is trained with the backpropagation method, which affects the weights of both the CNN-encoder and CNN-decoder. The loss function chosen should represent the reconstruction error, so that the training process will minimize it. In convolutional autoencoders, but also in every application of CNNs, it is common that input images have pixel values normalized in the interval [0,1], rather than [0, 255], since smaller values guarantee more stability. In this case, if the activation function of the last layer of the CNN-decoder is the sigmoid function, the output will be an image with the same dimensions of the input image, and values in the interval [0,1].

Although pixel values don't correspond to probabilities, we can use the Binary Cross

Entropy Loss function as the loss function to be minimized. If the value of the input image in position (x, y, z) is $r_{x,y,z}$ and the corresponding value of the reconstructed image is $\hat{r}_{x,y,z}$, BCE loss is:

$$L_{x,y,z}(r, \hat{r}) = -r_{x,y,z}\log(\hat{r}_{x,y,z}) - (1 - r_{x,y,z})\log(1 - \hat{r}_{x,y,z}) \tag{2.61}$$

We are now going to prove that L is a suitable loss function. In other words, we need to prove that L is minimized when $r = \hat{r}$ (which is the training goal). To this end, we calculate the derivative of L with respect to $\hat{r}$ and set it equal to zero:

$$\frac{\partial L}{\partial \hat{r}} = 0 \Leftrightarrow -r\frac{1}{\hat{r}} - (1 - r)(-\frac{1}{1 - \hat{r}}) = 0 \Leftrightarrow -r(1 - \hat{r}) + (1 - r)\hat{r} = 0 \Leftrightarrow \hat{r} = r \tag{2.62}$$

which means that L is minimized when the reconstructed image has same values as the original one.

2.5.4.3 Other types of autoencoders

We will now briefly refer to other variations of autoencoders with interesting properties.

Regularized Autoencoders As already stated above, to achieve a meaningful feature extraction from the input, we need the code dimension to be lower than the input dimension. Instead of limiting the code dimensions or the encoder and decoder capacity to ensure the extraction of meaningful features, we should be able to arrange them according to our problem and data distribution. This is achieved by regularized autoencoders, the most important of which are the sparse autoencoder and the denoising autoencoder.

Sparse autoencoders are typical autoencoders, but they add a sparsity penalty to the reconstruction loss. This is useful for learning features for another task, for example classification. Autoencoders regularized to be sparse take into account statistical properties of the training data set and not just learn an identity function, resulting in also learning useful features in parallel.

Denoising autoencoders use a different approach: instead of adding a sparsity penalty, they change the reconstruction error term of the loss function. While classical autoencoders try to minimize $L(\mathbf{x}, g(f(\mathbf{x})))$, the goal of denoising autoencoders is to minimize $L(\mathbf{x}, g(f(\tilde{\mathbf{x}})))$, where $\tilde{\mathbf{x}}$ is $\mathbf{x}$ corrupted with noise. So, in order to reconstruct the input, the autoencoder has to remove the noise and not just learn an identity function. In this way, it learns the structure of $p_{data}(\mathbf{x})$, thus also learning important properties of the data set as a byproduct.(Goodfellow et al. (2016))

Stochastic Autoencoders Stochastic autoencoders are a modern type of autoencoders that have expanded the idea of the encoder and the decoder beyond deterministic functions to stochastic mappings $p_{encoder}(\mathbf{h}|\mathbf{x})$ and $p_{decoder}(\mathbf{h}|\mathbf{x})$. (Goodfellow et al. (2016))

2.5.4.4 Applications

Autoencoders are powerful tools for dimensionality reduction or feature learning. Dimensionality reduction was one of the first applications of representation learning and deep learning in general. It was one of the early motivations for studying autoencoders. Lower dimensional representations can improve the performance on many different tasks, such as classification, since models of smaller spaces consume less memory and runtime.

Dimensionality reduction benefits especially information retrieval tasks (finding entries in a database), since search can become very efficient in some kinds of low-dimensional spaces.

Applications of undercomplete autoencoders include compression, recommendation systems as well as outlier detection.

Convolutional autoencoders are frequently used in image compression and denoising. They may also be used in image search applications, since the hidden representation often carries semantic meaning.

Recently, theoretical connections between autoencoders and latent variable models have brought autoencoders to the forefront of generative modeling. More specifically, the variational autoencoder is a generative model which can be trained and used to generate images. ((Goodfellow et al. 2016))

2.6 Feature pre-processing

2.6.1 Feature selection

Feature selection is an important stage of the system design pipeline (see Section 2.4) and it refers to choosing which features are the more relevant and important for the solution of our specific problem. In general, the curse of dimensionality is a problem associated with pattern recognition, since large representations require time and space resources, increase computational complexity and also reduce the performance of models trained on them. This happens because the more the features, the more the model parameters. When we have many model parameters, we also need a large data set of training samples, otherwise overfitting of the model on training data is often the case. So, our goal is to reduce features used and in the same time keep features which contain information that best discriminates patterns that belong to different classes and also guarantees close representations of patterns of the same class. In case of unsupervised learning, we can't use this criterion, so we resort to different solutions, such as the variance of the features.

2.6.1.1 Variance Thresholder

Variance Thresholder removes features, whose variance doesn't meet a certain threshold. Let's say we have a data set consisting of M N-dimensional feature vectors $\mathbf{x}_i$, $i = 1, ..., M$. So, each sample is described by N different features. Mean and variance estimators are used for the calculation of mean and variance (v) of every feature, respectively:

$$\bar{x}_j = \frac{1}{M} \sum_{i=1}^{i=M} x_{i,j} \tag{2.63}$$

$$\sigma_j^2 = \frac{1}{M} \sum_{i=1}^{i=M} (x_{i,j} - \bar{x}_j)^2 \tag{2.64}$$

for all $j = 1, ..., N$. Then, we keep only features x_j, where:

$$\sigma_j^2 > v_t \tag{2.65}$$

where v_t is a threshold set by the user, often taking into account how much dimensionality reduction we want.

2.6.2 Feature scaling

Feature scaling is important to ensure equal treatment for all features, regardless of their dynamic range. Its goal is to set all features values in a specific range, to prevent features with large values from having a greater impact on training a model than the others by

affecting the cost function more. In general, gradient descent methods are affected negatively by this problem, because the step size of the method will be greater for features with large values, and thus smooth transition towards the function minimum isn't guaranteed.

Feature scaling is also essential for algorithms that use distance metric between feature vectors. In case of big range differences between features, there is an imbalance in the amount of contribution each feature has in the distance calculations. If, for example, an algorithm tries to calculate the similarity between two feature vectors using their distance, features with large values will play a more important role. However, these features are not necessarily more significant for our task.

2.6.2.1 Normalization

A usual type of normalization scales features values in range [0,1]. This is achieved by changing every feature value according to the following equation:

$$x'_{i,j} = \frac{x_{i,j} - \min_{i=1,\dots,M} x_{i,j}}{\max_{i=1,\dots,M} x_{i,j} - \min_{i=1,\dots,M} x_{i,j}} \tag{2.66}$$

A benefit of this normalization method is that we always know the exact range in which feature values lie. On the other hand, it is quite problematic in case of outliers; if a few feature values are very large compared to the others, the rest will be squeezed in a small subinterval of [0,1].

2.6.2.2 Standardization

Another very popular scaling procedure is standardization, meaning that we transform feature values so that they have zero mean and unit variance. This is achieved by the following transform:

$$x'_{i,j} = \frac{x_{i,j} - \bar{x}_j}{\sigma_j} \tag{2.67}$$

where $\bar{x}_j = \frac{1}{M}\sum_{i=1}^{i=M} x_{i,j}$ and $\sigma_j = \sqrt{\frac{1}{M}\sum_{i=1}^{i=M}(x_{i,j} - \bar{x}_j)^2}$. This normalization approach is suitable in case some features have a large variance. In this case, they might affect the training process of a model more compared to the rest of the features. For many models, data with a Gaussian-like distribution are likely to improve their performance after standardization. Standardization is also beneficial in case we have negative values and isn't affected by outliers as much as normalization.

2.6.3 Dimensionality reduction

Although feature selection and scaling are important for all the aforementioned reasons, there are also plenty of cases where feature vectors contain redundant information. Our goal is to get compact and informative representations that contain all the useful properties of the data, while having the less possible dimensions. This is achieved by transforming the given features to a new set of less features, which will exhibit high information packing properties. This procedure is referred to as *dimensionality reduction*. There are many dimensionality reduction techniques, such as Principal Component Analysis, Independent Component Analysis, Singular Value Decomposition etc. Here, we will focus on Principal Component Analysis, which is one of the most widely used.

2.6.3.1 Principal Component Analysis (PCA)

PCA is a linear orthogonal transformation, whose goal is to give lower-dimensional data, while preserving as much of the data's variation as possible.

In general, a desirable property of features is to be mutually uncorrelated, in order to avoid information redunduncies. In this way, PCA is basically a transform that wants to ensure the newly generated features are uncorrelated. Let $\mathbf{x}$ denote a feature vector. We assume the data samples have zero mean for simplicity; otherwise, we just subtract the mean value. If we apply a linear transform on $\mathbf{x}$, we get $\mathbf{y}$:

$$\mathbf{y} = A^T\mathbf{x} \tag{2.68}$$

Since we have supposed that $E[\mathbf{x}] = 0$, then also $E[\mathbf{y}] = 0$. The correlation matrix of $\mathbf{y}$ is:

$$R_y = E[\mathbf{y}\mathbf{y}^T] = E[A^T\mathbf{x}\mathbf{x}^T A] = A^T R_x A \tag{2.69}$$

In practice, if we have a data set consisting of n feature vectors $\mathbf{x}_k$, $k = 1, ..., n$, the correlation matrix R_x is estimated as an average:

$$R_x = \frac{1}{n}\sum_{k=1}^{n} x_k x_k^T \tag{2.70}$$

We observe that R_x is a symmetric matrix. This means that it has mutually orthogonal eigenvectors. So, if we choose A so that its columns are the eigenvectors $\mathbf{a}_i$, $i = 0, ..., n-1$ of R_x, then it can be shown that R_y is diagonal, with the respective eigenvalues λ_i, $i = 0, ..., n-1$ of R_x on the diagonal. If we further assume that R_x is positive definite, then the eigenvalues are positive. So, the resulting features are indeed mutually uncorrelated, as our initial goal was.

We now need to examine how the dimension of these features can be reduced. In other words, we have to check which of the generated features are the most important to keep.

It can be shown that:

$$\mathbf{x} = \sum_{i=0}^{n-1} y(i)\mathbf{a}_i \tag{2.71}$$

$$y(i) = \mathbf{a}_i^T\mathbf{x} \tag{2.72}$$

If we define a new vector $\hat{\mathbf{x}}$ in the m-dimensional subspace

$$\hat{\mathbf{x}} = \sum_{i=0}^{m-1} y(i)\mathbf{a}_i \tag{2.73}$$

we get the projection of $\mathbf{x}$ onto a subspace spanned by the m orthonormal eigenvectors included in the sum defined above. We can now compare $\mathbf{x}$ and its projection by calculating their mean squared error:

$$E\left[\left\|\mathbf{x} - \hat{\mathbf{x}}\right\|^2\right] = E\left[\left\|\sum_{i=m}^{n-1} y(i)\mathbf{a}_i\right\|^2\right] \tag{2.74}$$

We now have to choose the eigenvectors that minimize this mean squared error. We have:

$$E\left[\left\|\sum_{i=m}^{n-1} y(i)\mathbf{a}_i\right\|^2\right] = E\left[\sum_i \sum_j y(i)\mathbf{a}_i^T y(j)\mathbf{a}_i\right] \tag{2.75}$$

Since the eigenvectors $\mathbf{a}_i$, $i = 1, ..., n-1$ are orthonormal:

$$\mathbf{a}_i^T \mathbf{a}_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \tag{2.76}$$

So, equation 2.75, taking into account equations 2.76 and 2.72 becomes:

$$E\left[\sum_i \sum_j y(i)\mathbf{a}_i^T y(j)\mathbf{a}_i\right] = \sum_{i=m}^{n-1} E[y^2(i)] = \sum_{i=m}^{n-1} \mathbf{a}_i^T E[\mathbf{x}\mathbf{x}^T]\mathbf{a}_i = \sum_{i=m}^{n-1} \mathbf{a}_i^T R_x \mathbf{a}_i \tag{2.77}$$

Taking into account the definition of eigenvectors and equation 2.74:

$$E\left[\left\|\mathbf{x} - \hat{\mathbf{x}}\right\|^2\right] = \sum_{i=m}^{n-1} \mathbf{a}_i^T \lambda_i \mathbf{a}_i = \sum_{i=m}^{n-1} \lambda_i \tag{2.78}$$

It is now obvious that, if for the new vector $\hat{\mathbf{x}}$, defined in equation 2.73, we choose the eigenvectors that correspond to the m largest eigenvalues, the MSE is minimized, since it is equal to the sum of the $n - m$ smallest eigenvalues. That's why the method is called Principal Component Analysis, since we use the principal components to determine the new feature vectors.

PCA has another interesting property. More specifically, if $E[\mathbf{x}] = 0$ and $\mathbf{y}$ is the transformed feature vector after PCA, then the variance of each component of $\mathbf{y}$ is $\sigma_{y_i^2} = E[y^2(i)] = \lambda_i$. So, the eigenvalues of the correlation matrix of the initial features are equal to the variances of the transformed features. This means that, since we choose features which correspond to maximum eigenvalues, the sum of the eigenvalues, and as a result their variances, is maximized. Thus, the selected m features preserve most of the total variance associated with the original features. (Theodoridis and Koutroumbas (2009))

2.7 Clustering

We first referred to clustering in Section 2.2.2 as an unsupervised learning method, whose goal is to create groups of patterns with similar features. In clustering problems, we don't know the labels or target values of each pattern; that's why we want to create an organization of patterns in groups (clusters), which will reveal similarities and differences between them and will also give us useful information about their structure.

2.7.1 Definition of clustering

Let's suppose we have a data set X:

$$X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N] \tag{2.79}$$

We define the partition of X into m sets (clusters) $C_1, C_2, ..., C_m$ as m-clustering, when the following conditions are met:

1. $C_i \neq \emptyset$, $i = 1, ..., m$

2. $\cup_{i=1}^{m} C_i = X$

3. $C_i \cap C_j = \emptyset,\ i \neq j,\ i, j = 1, ..., m$

It is important that vectors that belong to cluster C_i are more similar to each other and less similar to feature vectors that belong to other clusters. This similarity is defined by a distance measure, depending on the application and the algorithmic approach we choose for deriving these clusters.

According to condition 3, every feature vector can belong to only one cluster. This clustering approach is called *hard* clustering. If we employ a fuzzy approach to clustering, each feature vector is accompanied by a membership function, quantifying the degree with which the vector belongs to each cluster (*soft* clustering).

There are many different categories of clustering algorithms, i.e. learning procedures whose goal is to identify the characteristics and properties that specify the clusters underlying the data set. We will refer to the most common categories:

- Sequential algorithms. These algorithms are fast and produce a single clustering. The feature vectors are presented to the algorithm several times and compact clusters (often hyperspherical or hyperellipsoid) are usually produced, according to the distance metric employed. Most of the times, the final result depends on the order in which feature vectors are presented to the algorithm.

- Hierarchical algorithms. These are further divided into two categories:

  - Agglomerative algorithms. These algorithms produce a sequence of clusterings consisting of a decreasing number of clusters gradually, starting from m different clusters and merging two resulting clusters at each step.
  - Divisive algorithms. These algorithms act in the opposite way; they increase the number of clusters at each step by splitting one cluster of the previous step.

- Clustering algorithms based on cost function optimization. These algorithms are, in general, iterative and produce successive clusterings in order to minimize a cost function J, which is chosen to evaluate the clusterings. Usually, they use a fixed number of clusters m and they terminate when a local optimum of J is reached.

We are going to go through some of the most popular clustering algorithms; K-Means, Gaussian Mixture Models and Agglomerative Clustering.

2.7.1.1 K-Means clustering

K-Means is one of the most known and used clustering algorithms. It belongs to the third category mentioned above, i.e. it's a clustering algorithm based on the optimization of a cost function. Let's suppose we have feature vectors $\boldsymbol{x}_i,\ i = 1, ..., n$ and cluster representatives $\boldsymbol{\theta}_j,\ j = 1, ..., m$. Each vector is interpreted as a point representative and the dissimilarity between vectors and cluster representatives is calculated using the squared Euclidean distance.

This means that the cost function is defined as:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \left\| \mathbf{x}_i - \boldsymbol{\theta}_j \right\|^2 \tag{2.80}$$

The cluster representative $\boldsymbol{\theta}_j$ is the mean vector of the jth cluster. The algorithm is presented below:

- We first choose initial arbitrary cluster representatives $\boldsymbol{\theta}_j(0)$, $j = 1, ..., m$.
- We repeat:
  - For $i = 1, ...n$
    - Determine which $\boldsymbol{\theta}_j$ is closer to $\boldsymbol{x}_i$ according to the euclidean distance:

$$\arg\min_j \left\|\boldsymbol{x}_i - \boldsymbol{\theta}_j\right\|^2 \tag{2.81}$$

    - Set $b(i) = j$, to keep the information of the cluster $\boldsymbol{x}_i$ is assigned to.
  - End {For}
  - For $j = 1, ..., m$
    - We update the parameters, i.e. $\boldsymbol{\theta}_j$, as the mean of vectors $\boldsymbol{x}_i$ which have been assigned to cluster j (i.e. $b(i) = j$).
  - End {For}.
- We repeat the procedure until there is no change in $\boldsymbol{\theta}_j$ for every $j = 1, ..., m$.

The algorithm converges and gives compact clusters. It is important to observe that there is a dependence of the final clustering on the initialization of the K-means centroids $\boldsymbol{\theta}_j$, though if the features are well selected, this dependence is restricted.

Although K-Means is a very popular algorithm, and easy to understand and implement, it has some limitations. Since the clusters' centroids are updated using the mean value, the resulting clusters have a circular shape. However, this is not always the case; the data distribution doesn't necessarily have a circular form. Also, the time complexity of K-Means is $O(n^2)$, which isn't very suitable for large applications. The next algorithm we are going to present is distribution-based, rather than distance-based. (Theodoridis and Koutroumbas (2009))

A modification of typical K-Means algorithm, called *Mini-Batch K-Means*, uses subsets of randomly selected samples (mini-batches) in each training iteration to reduce the amount of computation needed for convergence of the algorithm.

2.7.1.2 Gaussian Mixture Models

This is also a clustering algorithm that optimizes a cost function. In general, it belongs to the family of Mixture Decomposition Schemes, an algorithmic family based on the Bayesian philosophy. We assume that there are m distinct clusters C_j, $j = 1, ..., m$ underlying the data set. Each feature vector $\boldsymbol{x}_i$ belongs to a cluster C_j with probability $P(C_j|\boldsymbol{x}_i)$. In this sense, we can state that each point $\boldsymbol{x}_i$ can be derived from any of the j distributions with probability $P(C_j)$:

$$p(\boldsymbol{x}) = \sum_{j=1}^{m} p(\boldsymbol{x}|C_j)P(C_j) \tag{2.82}$$

where $\sum_{j=1}^{m} P(C_j) = 1$ and $\int_{\boldsymbol{x}} p(\boldsymbol{x}|C_j)d\boldsymbol{x} = 1$. It can be shown that this model can approach any continuous probability distribution, if the number of mixture components m is appropriate and suitable parameters are chosen.

A feature vector $\boldsymbol{x}_i$ is appointed to cluster C_j if

$$P(C_j|\boldsymbol{x}_i) > P(C_k|\boldsymbol{x}_i), \quad k = 1, ..., m, \; k \neq j \tag{2.83}$$

So, our goal is to calculate all $P(C_k|\boldsymbol{x}_i)$, $k = 1, ..., m$ and find $j_i = \arg\max_k P(C_k|\boldsymbol{x}_i)$, for every $i = 1, ..., n$. Using the Bayes probability rule:

$$P(C_j|\boldsymbol{x}_i) = \frac{p(\boldsymbol{x}_i|C_j)P(C_j)}{p(\boldsymbol{x}_i)} = \frac{p(\boldsymbol{x}_i|C_j)P(C_j)}{\sum_{j=1}^{m} p(\boldsymbol{x}_i|C_j)P(C_j)} \tag{2.84}$$

This means that we basically need to calculate $p(\boldsymbol{x}_i|C_j)P(C_j)$ for every $j = 1, ..., m$. The first step is to choose a set of $p(\boldsymbol{x}_i|C_j)$ in parametrical form, i.e. $p(\boldsymbol{x}_i|C_j; \boldsymbol{\theta})$ and then we have to use an algorithm to calculate both $\boldsymbol{\theta}$ and $P(C_j)$, $j = 1, ..., m$. For the solution to this problem we need to employ iterative algorithms, since we have no information about the mixture component (or cluster) from which each sample comes.

So, our initial full data set consists of pairs $\boldsymbol{y} =(x_i, C_i)$, $i = 1, ..., n$, where C_i is to be found. The vector of unknown parameters $\boldsymbol{\Theta}^T$ consists of $\boldsymbol{\theta}$ and the unknown probabilities $\boldsymbol{P} = [P(C_1), P(C_2), ..., P(C_m)]$:

$$\boldsymbol{\Theta}^T = [\boldsymbol{\theta}^T, \mathbf{P}^T]^T \tag{2.85}$$

In this kind of problems, where C_l is unknown, we can't just calculate $\boldsymbol{\theta}$ with Maximum Likelihood Estimation (MLE). More specifically, if we assume that $\boldsymbol{y}_i$, $i = 1, ..., N$ are independent, the goal of MLE is to maximize $\prod_{i=1}^{n} p(\boldsymbol{y}_i; \boldsymbol{\theta})$, or $\sum_{i=1}^{n} \ln(p(\boldsymbol{y}_i; \boldsymbol{\theta}))$ so, the MLE of $\boldsymbol{\theta}$ is:

$$\hat{\boldsymbol{\theta}} = \sum_{i=1}^{n} \frac{\partial \ln(p(\mathbf{y}_i; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} \tag{2.86}$$

However, we don't know the full observations $\boldsymbol{y}_i$, which means that now we don't know the exact $p(\boldsymbol{y}_i; \boldsymbol{\theta})$. That's why we are going to maximize the expected value of $p(\boldsymbol{y}_i; \boldsymbol{\theta})$, with an iterative algorithm called Expectation Maximization. We first initialize the parameters $\boldsymbol{\Theta}(0)$. Then, each iteration of the algorithm consists of 2 steps.

In the Expectation Step $t + 1$ we calculate the expected value of the log-probability (assuming that $\boldsymbol{y}_i$ are independent and taking into account $\boldsymbol{\Theta}(t)$):

$$\begin{aligned} Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}(t)) &= E\big[\sum_{i=1}^{n} \ln(p(\boldsymbol{x}_i, C_j; \boldsymbol{\theta}))|X, \boldsymbol{\Theta}(t)\big] \\ &= \sum_{i=1}^{n} E\big[\ln(p(\boldsymbol{x}_i, C_j; \boldsymbol{\theta})|X, \boldsymbol{\Theta}(t))\big] \qquad (2.87) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m} P(C_j|\boldsymbol{x}_i; \boldsymbol{\Theta}(t))\ln(p(\boldsymbol{x}_i, C_j; \boldsymbol{\theta})) \qquad (2.88) \end{aligned}$$

Since

$$p(\boldsymbol{x}_i, C_j; \boldsymbol{\theta}) = p(\boldsymbol{x}_i|C_j; \boldsymbol{\theta})P(C_j) \tag{2.89}$$

Equation 2.88 can be written as:

$$Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}(t)) = \sum_{i=1}^{n}\sum_{j=1}^{m} P(C_j|\boldsymbol{x}_i; \boldsymbol{\Theta}(t))\ln(p(\boldsymbol{x}_i|C_j; \boldsymbol{\theta})P(C_j))$$

Now, in our case, we want to derive m Gaussian mixture components, where each one has parameters $\boldsymbol{\mu}_j, \Sigma_j$, where Σ_j is assumed to be a diagonal covariance matrix $\Sigma_j = \sigma_j^2 I$

for simplicity reasons. So, $\boldsymbol{\theta}_j = [\boldsymbol{\mu}_j, \sigma_j]$ and the probability distribution $p(\mathbf{x}_i|C_j;\boldsymbol{\theta})$ is defined as:

$$p(\boldsymbol{x}_i|C_j;\boldsymbol{\theta}) = \frac{1}{(2\pi\sigma_j^2)^{l/2}}\exp\Bigg(-\frac{\big\|\boldsymbol{x}_i-\boldsymbol{\mu}_j\big\|^2}{2\sigma_j^2}\Bigg) \tag{2.90}$$

So, now the final equation of the Expectation step can be written as:

$$Q(\boldsymbol{\Theta};\boldsymbol{\Theta}(t)) = \sum_{i=1}^{n}\sum_{j=1}^{m} P(C_j|\boldsymbol{x}_i;\boldsymbol{\Theta}(t))\Bigg(-\frac{l}{2}\ln\sigma_j^2 - \frac{1}{2\sigma_j^2}\big\|\boldsymbol{x}_i-\boldsymbol{\mu}_j\big\|^2 + \ln P(C_j)\Bigg) \tag{2.91}$$

We are now proceeding to the Maximization Step, whose goal is to maximize $Q(\boldsymbol{\Theta};\boldsymbol{\Theta}(t))$ with respect to $\boldsymbol{\mu}_j$, σ_j^2 and $P(C_j)$. This means that $\boldsymbol{\mu}_j(t+1)$ is given by:

$$\frac{\partial Q(\boldsymbol{\Theta};\boldsymbol{\Theta}(t))}{\partial \boldsymbol{\mu}_j} = 0 \tag{2.92}$$

which results in:

$$\boldsymbol{\mu}_j(t+1) = \frac{\sum_{i=1}^{n} P(C_j|\mathbf{x}_i;\boldsymbol{\Theta}(t))\boldsymbol{x}_i}{\sum_{i=1}^{n} P(C_j|\mathbf{x}_i;\boldsymbol{\Theta}(t))} \tag{2.93}$$

In a similar way, we have:

$$\sigma_j^2(t+1) = \frac{\sum_{i=1}^{n} P(C_j|\boldsymbol{x}_i;\boldsymbol{\Theta}(t))\big\|\boldsymbol{x}_i-\boldsymbol{\mu}_j(t+1)\big\|^2}{\sum_{i=1}^{n} P(C_j|\boldsymbol{x}_i;\boldsymbol{\Theta}(t))} \tag{2.94}$$

$$P(C_j)(t+1) = \frac{1}{n}\sum_{i=1}^{n} P(C_j|\boldsymbol{x}_i;\boldsymbol{\Theta}(t)) \tag{2.95}$$

where

$$P(C_j|\boldsymbol{x}_i;\boldsymbol{\Theta}(t)) = \frac{p(\boldsymbol{x}_i|C_j;\boldsymbol{\theta}(t))P(C_j)(t)}{p(\boldsymbol{x}_i;\boldsymbol{\Theta}(t))} \tag{2.96}$$

$$p(\boldsymbol{x}_i;\boldsymbol{\Theta}(t)) = \sum_{j=1}^{m} p(\boldsymbol{x}_i|C_j;\boldsymbol{\theta}(t))P(C_j)(t) \tag{2.97}$$

The algorithm terminates when $\big\|\boldsymbol{\Theta}(t+1)-\boldsymbol{\Theta}(t)\big\| \leq \epsilon$, where $\|\cdot\|$ is an appropriate norm and ϵ a small user-defined constant.

After the last iteration, we will now know $P(C_j|\boldsymbol{x}_i)$ for each i and j, so vectors are assigned to clusters according to this probability.

This algorithm takes into account not only the mean, but also the variance of the data, which makes it more flexible in the shape of data they can group. It can also be interpreted as a soft clustering algorithm, since for each vector, we have a probability of it belonging to each cluster. These are two important advantages, compared to K-Means. On the other hand, it is more complex to understand and to implement than K-Means. As far as time complexity is concerned, it is $O(nmD^3)$, where D is the problem dimension. This means that GMM can't be efficiently used for high-dimensional tasks. (Theodoridis and Koutroumbas (2009))

2.7.1.3 Agglomerative Clustering

Agglomerative clustering is a type of Hierarchical clustering. Hierarchical clustering algorithms have a different philosophy compared to the clustering methods we have already presented. They produce a sequence of nested clusterings, instead of a single cluster. A clustering $\Re_1$ containing k clusters is said to be *nested* in the clustering $\Re_2$ containing $r < k$ clusters if every cluster in $\Re_1$ is a subset of a set in $\Re_2$. At least one cluster of $\Re_1$ is a proper subset of $\Re_2$, in which case we write $\Re_1 \sqsubset \Re_2$.

In agglomerative algorithms, the initial clustering $\Re_0$ consists of n clusters, each containing a single vector $\boldsymbol{x}_i$. At the first step, $\Re_1$ is produced, for which $\Re_0 \sqsubset \Re_1$ holds, by merging two of $\Re_0$'s clusters. The algorithm proceeds until a unique cluster is produced, which contains all feature vectors $\boldsymbol{x}_i$. If we define the desired number of clusters, m, the algorithm terminates after $n - m$ steps (of course $m \leq n$).

We are now going to present the general agglomerative scheme. We define as $g(C_j, C_k)$ a function for all pairs of clusters, measuring the proximity between C_j and C_k. For example, it is common that the minimizing of variance of merged clusters is chosen as the proximity criterion, which means that g is the function that calculates variance (dissimilarity function).

More specifically, the algorithm is presented below, where t denotes the level of hierarchy:

- Initialization:
  - Choose $\Re_0 = \{C_i = \boldsymbol{x}_i i = 1, ..., n\}$ as the initial clustering.
  - $t = 0$
- Repeat:
  - $t = t + 1$
  - Find the pair of clusters C_j, C_k among all pairs C_r, C_s of $\Re_{t-1}$ such that:

$$g(C_j, C_k) = \begin{cases} \min_{r,s} g(C_r, C_s), & \text{if } g \text{ is a dissimilarity function} \\ \max_{r,s} g(C_r, C_s), & \text{if } g \text{ is a similarity function} \end{cases} \tag{2.98}$$

  - Define the new cluster $C_q = C_j \cup C_k$ and update the clustering $\Re_t = (\Re_{t-1} - \{C_j, C_k\}) \cup \{C_q\}$.
- Until all vectors lie in the same cluster, or a number of clusters m is predefined, in which case until we reach $n - m$ steps.

A disadvantage of nested clustering is that in case a "poor" cluster has occurred in an early stage, it can't be corrected or improved in the next steps. Time complexity of a naive implementation of this algorithm is $O(n^3)$, though using the most suitable data structures and other optimizations can reduce the complexity to $O(n^2)$. An advantage of this algorithm compared to K-Means and GMM is that it doesn't require a random initialization, like the other two, and thus it results in a deterministic clustering. (Theodoridis and Koutroumbas (2009))

2.7.1.4 Deep Embedded Clustering

Deep Embedded Clustering is a relatively new approach to clustering, introduced by Xie et al. (2016). Its goal is to efficiently produce meaningful clusters and simultaneously improve the pattern representations used for the clustering, using an iterative scheme. We define the problem as follows:

We have a set of n points x_i, $\{x_i \in X\}_{i=1}^{n}$ that we want to organize in k clusters. Each cluster is represented by a centroid μ_j, $j = 1, ..., k$. Instead of clustering the points directly in feature space X, we transform the feature vectors to a latent space Z using the mapping $f_\theta : X \to Z$, where θ are the learnable parameters.

This mapping is successfully learned and applied by an autoencoder (see Section 2.5.4). So, DEC basically consists of the following two steps:

1. Initialize the parameters and clusters centroids by training the autoencoder
2. Optimize the parameters and clusters centroids by iterating between computing an auxiliary target distribution and minimizing the Kullback-Leibler Divergence.

Step 1 is quite straightforward; the training of an autoencoder involves minimizing a reconstruction loss function. In this way, the parameters θ of the autoencoder are initialized. Then, we use the derived representations and the K-Means algorithm to generate a clustering and get the cluster centroids μ_j, $j = 1, ..., k$.

Step 2 is basically an iterative algorithm which alternates between two steps:

1. We want to create a soft assignment of each pattern (more specifically, each embedded point) to the available clusters. To achieve this, we have to use a similarity metric. The core of this metric is to use the family of Student's t distribution, so the assignment is done based on:

$$q_{ij} = \frac{(1 + \left\|z_i - \mu_j\right\|^2)^{-1}}{\sum_{j'}(1 + \left\|z_i - \mu_{j'}\right\|^2)^{-1}} \tag{2.99}$$

   where $z_i = f_\theta(x_i) \in Z$ is the embedded representation of x_i. It is obvious that q_{ij} can be interpreted as the probability of assigning pattern i to cluster j. This makes sense because, if the embedded representation z_i is close to cluster centroid μ_j, their Euclidean distance will be small and thus the numerator of q_{ij} will be large, indicating that pattern i belonging to cluster j is very likely.

2. We now want to refine the pattern latent representations and the resulting clusters by minimizing a Kullback-Leibler Divergence loss function in order to learn from high confidence assignments. For this purpose we need two distributions; the pattern distribution, modeled with q, and a target distribution. So, our loss function between our assignments q_i and an auxiliary target distribution p_i is:

$$L = KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.100}$$

   The last pending issue is the selection of distribution p_i, which should satisfy some desired properties. It would be beneficial if it could improve cluster purity by strengthening the predictions and emphasize mainly the points that are assigned with high confidence. Taking these into consideration, the proposed target distribution (Xie et al. (2016)) is:

$$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j'} q_{ij'}^2 / f_{j'}} \tag{2.101}$$

   where $f_j = \sum_i q_{ij}$ are soft clusters frequencies. This choice of KL Divergence and p distribution is essential, since it is empirically shown that they guarantee a greater contribution of points closest to the cluster centers (with higher q_{ij}) to the gradient $\frac{\partial L}{\partial z_i}$, thus ensuring an emphasis on high confidence predictions. (Xie et al. (2016))

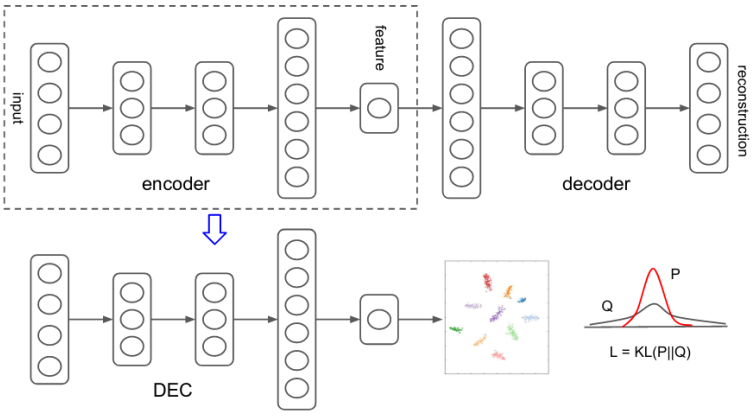


Figure 2.13: Deep Embedded Clustering algorithm. (Xie et al. (2016))

For the optimization of the parameters θ and the cluster centroids μ_j we use a gradient descent procedure (see Section 2.3.2). The gradients $\frac{\partial L}{\partial z_i}$ are backpropagated to update the weights of the autoencoder network, whereas centroids μ_j are updated using $\frac{\partial L}{\partial \mu_j}$. The iterative scheme described is terminated when the percentage of cluster assignments that change in each iteration is below a certain threshold.

2.7.1.5 Semi-supervised Deep Embedded Clustering

Up to now, we have presented a completely unsupervised clustering procedure. Now, the goal is to further improve the resulting clustering by using human intervention in the training procedure. One way of doing so, is to use partially annotated data in the form of pairwise constraints. This means that we have some information about pairs of patterns which are manually annotated as to whether or not they should belong to the same cluster. These pairwise constraints can lead the direction of clustering and embedding. (Ren et al. (2018))

For storing these constraints, we employ a square matrix A (NxN), where N is the number of training samples. This matrix contains must-link and cannot-link connections between patterns. Must-link means that the two patterns should be assigned to the same cluster, whereas cannot-link means that the two patterns should be in separate clusters. For each element a_{ij} we have:

$$a_{ij} = \begin{cases} 1, & \text{if } i-j \text{ must-link} \\ -1, & \text{if } i-j \text{ cannot-link} \\ 0, & \text{if we have no information about } i-j \end{cases} \tag{2.102}$$

This information has to be integrated in the loss function, which will now consist of two terms: the KL Divergence and a term regarding the pairwise constraints. The second term is set bearing in mind that two patterns which are supposed to belong to the same cluster should have close representations in the latent space, in contrast to cannot-link pairs, whose representations should lie further away. The new loss function is:

$$L = KL(P\|Q) + \lambda \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} \|z_i - z_j\|^2 \tag{2.103}$$

where λ is a user defined constant which indicates how much impact the second term will have on loss computations. This term can be interpreted as follows: If two patterns i and j are connected with a must-link constraint ($a_{ij} = 1$), z_i should be close to z_j; otherwise, L

will increase. On the other hand, if i and j are connected with a cannot-link constraint ($a_{ij} = -1$), z_i should be far from z_j, so that the loss can be significantly reduced.

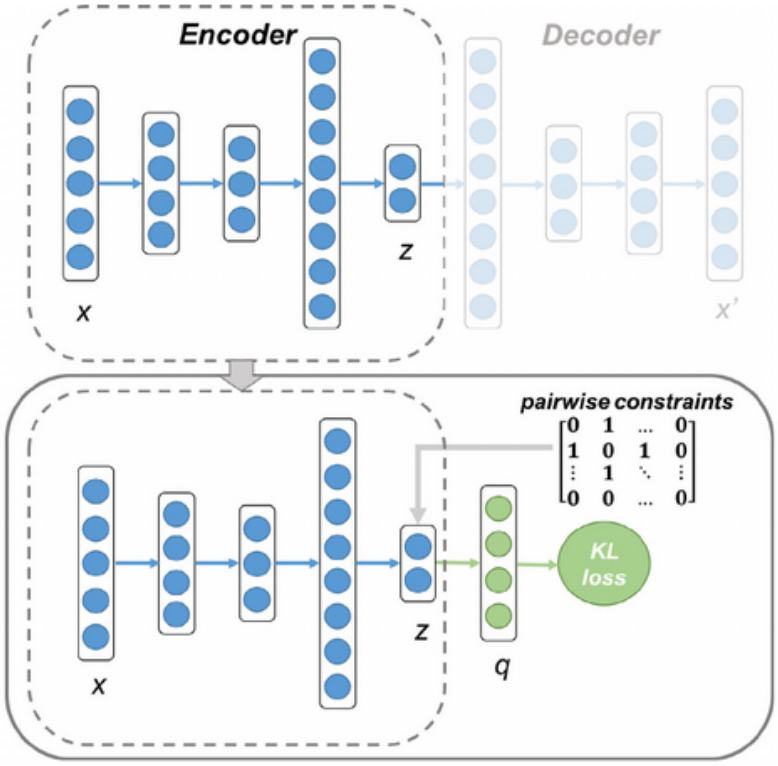


Figure 2.14: Semi-Supervised Deep Embedded Clustering algorithm. (Ren et al. (2018))

The backpropagation of the loss in the network and the cluster centroids updates have no difference compared to DEC. (Ren et al. (2018))

2.8 Evaluation Metrics

In this section we will go through some useful evaluation metrics used in the next chapters.

2.8.1 Confusion Matrix

Confusion Matrix is very helpful in classification evaluation tasks. In particular, it examines the confusion that might occur between classes, that's why it's called Confusion Matrix. It is a square matrix NxN, where N is the number of classes of the classification problem.

Each element of the matrix $A[i, j]$ contains the number of samples whose true class is i, but they were assigned to class j. For example, suppose that we have a two-class classification problem, and as a result a 2x2 confusion matrix (see Figure 2.15). TP stands for true positive and TN for true negative. It is obvious that elements of the diagonal of the matrix correspond to the right model predictions, whereas the rest of the elements to wrong predictions. FP stands for false positive predictions and FN for false negative.

The most important metrics calculated from this matrix are *recall*, *precision* and *accuracy*.

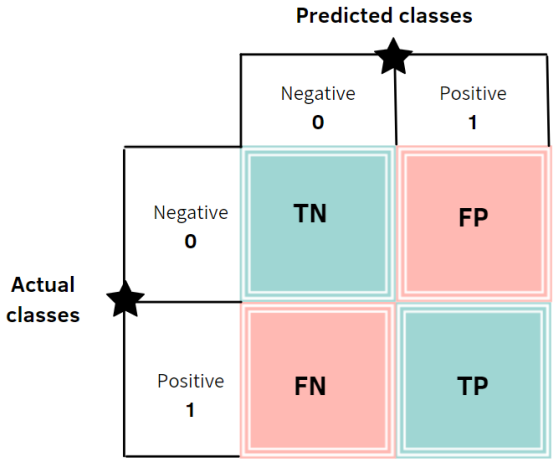


Figure 2.15: Confusion matrix for a binary classification problem.

Recall Recall R_i of class i is the percentage of patterns of class i that were actually predicted to belong to class i. For a multiclass classification problem:

$$R_i = \frac{A[i,i]}{\sum_{j=1}^{N} A[i,j]} \tag{2.104}$$

In the special case of 2 classes, $R_1 = \frac{TP}{TP+FN}$. Recall provides a measure of missed positive predictions, and thus of the coverage of the positive class. It is a suitable metric when false negatives are associated with a high cost (for example, in case of patient diagnosis, false negatives are dangerous for the health of the patients).

Precision Precision P_i of class i is the percentage of all patterns that are predicted to belong to class i that are actually of class i. For a multiclass classification problem:

$$P_i = \frac{A[i,i]}{\sum_{j=1}^{N} A[j,i]} \tag{2.105}$$

In the special case of 2 classes, $P_1 = \frac{TP}{TP+FP}$. Precision is a measure of the correct positive predictions that were made, so it is an appropriate measure for cases where false positives are highly unwanted (for example in spam detection, if an e-mail is falsely classified as spam, the user might lose important e-mails).

Precision and Recall are metrics that focus on different sides of the problem; our goal is to generate a metric that provides a more general evaluation of the classification task. This is partially achieved by accuracy.

Accuracy Accuracy Ac is the percentage of correctly classified patterns. For a multiclass classification problem:

$$Ac = \frac{\sum_{i=1}^{N} A[i,i]}{N} \tag{2.106}$$

In the special case of 2 classes, $Ac = \frac{TP+TN}{N}$.

Though Accuracy takes into account both false positives and false negatives, it is also significantly determined by true negatives, which in the majority of the tasks aren't particularly important. As a result, we often use the *F1-score* of precision and recall, which is their harmonic mean:

$$F1 = \frac{2 \cdot P \cdot R}{P + R} \tag{2.107}$$

In case of multi-class classification problems, F1 can be calculated for each class, and then the result is averaged. An issue arising in these cases is which average metric we should employ.

- *Micro F1 score*: We calculate the total TP, FP and FN and then directly calculate the F1 score. This approach doesn't favor any particular class.

- *Weighted F1 score*: For each class i, we calculate the F1 score F_i and then their weighted average, where each weight w_i depends on the number of true labels of each class:

$$F1 = \sum_{i=1}^{N} w_i F_i \tag{2.108}$$

  where N is the number of classes. As a result, this approach favors the class with the most samples.

- *Macro F1 score*: We calculate the F1 score F_i for each of the N classes and then just their classical average:

$$F1 = \sum_{i=1}^{N} F_i \tag{2.109}$$

  In this case, minority and majority classes are equally represented, which makes this approach more suitable in problems with highly imbalanced datasets.

(Theodoridis and Koutroumbas (2009))

2.8.2 Metrics for time segments evaluation

The confusion matrix and the aforementioned metrics can also be used for the evaluation of tasks that aren't clearly classification problems. In our case, if we want to effectively and accurately compare events detected in a signal by our algorithms to the actual events (ground truth), we have to find a way to adjust the confusion matrix idea to our problem, which is a time segment comparison. More specifically, each event is described by a start and end time; so, for a more complete approach, we do this evaluation on a temporal and on an event basis.

2.8.2.1 Temporal evaluation

Temporal evaluation refers to per-time unit comparison of time segments. We split the segments to discrete time units, which are chosen depending on the problem and the resolution we want to examine. We interpret the time segments comparison as a classification task of each time unit into two categories (event or no event). Then we calculate the precision and recall rates by comparing each unit of the detected events to the ground-truth events. Their harmonic mean is the temporal F1 score.

2.8.2.2 Event evaluation

To compare detected to ground truth events, we will again fill a 2x2 confusion matrix, though without true negatives (TN).

- True positives (TP) are the detected events whose time interval overlapped with one actual event.
- False positives (FP) are the detected events that did *not* have an overlap with an actual event (falsely detected).
- False negatives (FN) are the actual events that weren't detected (missed), i.e. didn't have an overlap with a detected event.

We then calculate precision and recall and their F1-score (event F1-score).

Another evaluation approach is to measure the event miss rate and the false discovery rate, defined as:

$$\text{miss rate} = \frac{FN}{TP + FN} = 1 - R \tag{2.110}$$

$$\text{false discovery rate} = \frac{FP}{TP + FP} = 1 - P \tag{2.111}$$

Chapter 3

Vocalization Detection

Chapter 2 provided fundamental information, necessary for understanding the core of our methods in both signal processing and machine learning. We are now going to focus on AMVOC's functionalities, starting from the first step to AMVOC's processing pipeline, which is to detect the time segments that contain a vocalization. This is achieved by inspecting and processing information from the spectrogram of the signal. The major difficulty regarding this task is to discriminate between *actual* vocalizations and noise (see Figure 3.1a and 3.1b).

(a) An actual vocalization

(b) Noise.

Figure 3.1: Examples of two different segments of the spectrogram, from 30-110 kHz. Here, an actual vocalization (a) and a noisy, high-energy segment (b) are displayed.

We have developed two separate tools for vocalization detection, the first one working on offline mode and the second one on online mode. Basically, offline mode means that we get a pre-existing recording and process it to determine the time segments corresponding to vocalizations, whereas in online mode the recording and detection procedures take place simultaneously.

3.1 Offline USV Detection

In order to detect the mice USVs, we first compute the spectrogram of the whole recording. This is done by splitting the signal to non-overlapping short-term windows (frames) of duration $w = 2$ ms (time resolution) and calculating the Short-Term Fourier Transform (STFT) for each time frame. Frequency resolution f_r is calculated as:

$$f_r = \frac{1}{w} \tag{3.1}$$

This means that in our case, if $w = 2$ ms, $f_r = 0.5$ kHz. In simple terms, the ability of the 2-dimensional (time & frequency) spectrogram representation to discriminate between different frequency coefficients is 0.5 KHz (i.e. frequency resolution). This is less fine-scaled than is typical for human speech analytics methods, but because mice vocalizations usually range in the frequencies 30-110 KHz, this resolution is more than sufficient.

As soon as the spectrogram is extracted, the USVs are detected on a time frame-basis, using two separate criteria, time-based thresholding (TT) and frequency-based thresholding (FT), that take into account the values of the distribution of the signal's energy at the different frequencies (Figure 3.2). Both of these criteria are based on the energies, however they differ in the way the thresholding criteria are calculated and applied. The details of the two criteria are as follows:

- Time-based thresholding (TT): This involves a simple temporal thresholding of the spectral energy values. To do this, for each time frame, we calculate the spectral energy by summing the spectrogram values at each frequency. We do this procedure for the frequency range of interest, which is, as mentioned above, from 30 kHz to 110 kHz. If we denote the spectrogram value at time frame i and frequency j as E_{ij}, spectral energy S_i is calculated as:

$$S_i = \sum_{j=30kHz}^{110kHz} E_{ij} \tag{3.2}$$

  where the step of j is equal to 0.5 kHz. We then compute a dynamic sequence of thresholds for the spectral energy. In particular, for each frame i, for which we have extracted the spectral energy S_i, we compute the dynamic threshold:

$$T_i = \frac{1}{2}\frac{\sum_{j=0}^{N-1} S_j}{N} + \frac{1}{2}\frac{\sum_{j=0}^{K-1} S_{i-j}}{K} \tag{3.3}$$

  where N is the number of time frames, so the first term refers to the mean spectral energy. In the second term, K is the size of a moving average filter in seconds. Here we use $K = 2$ seconds, which is convolved with the sequence of spectral energy values. In other words, the dynamic threshold T is defined at each frame i as the average of the current spectral energy (S_i) and the moving average of the spectral energies of the last K frames.

- Frequency-based thresholding (FT): This second criterion is associated to applying a thresholding rule, based on the per-frame distribution of energies on the different frequencies. A simple dynamic threshold at each time frame of the spectrogram (as described above) is not enough, because there are also time frames where high spectral energy occurs due to noise. The spectral energy value in these high-noise time frames may surpass the threshold, but this does not correspond to any vocalization (Figure 3.1a and 3.1b). Our goal is to filter out these false positive vocalizations. For example, in Figure 3.1b, in the noisy segment each time frame surpasses the spectral energy threshold (criterion 1), but not the second applied threshold (criterion 2). It is easy to observe that in the vast majority of cases, noise appears as high energy values, spread across a large frequency range in each time frame (Figure 3.1b), compared to the time frame energy distribution in vocalizations, which is concentrated in a small frequency range in each time frame (Figure 3.1a). Our filtering criterion was to keep

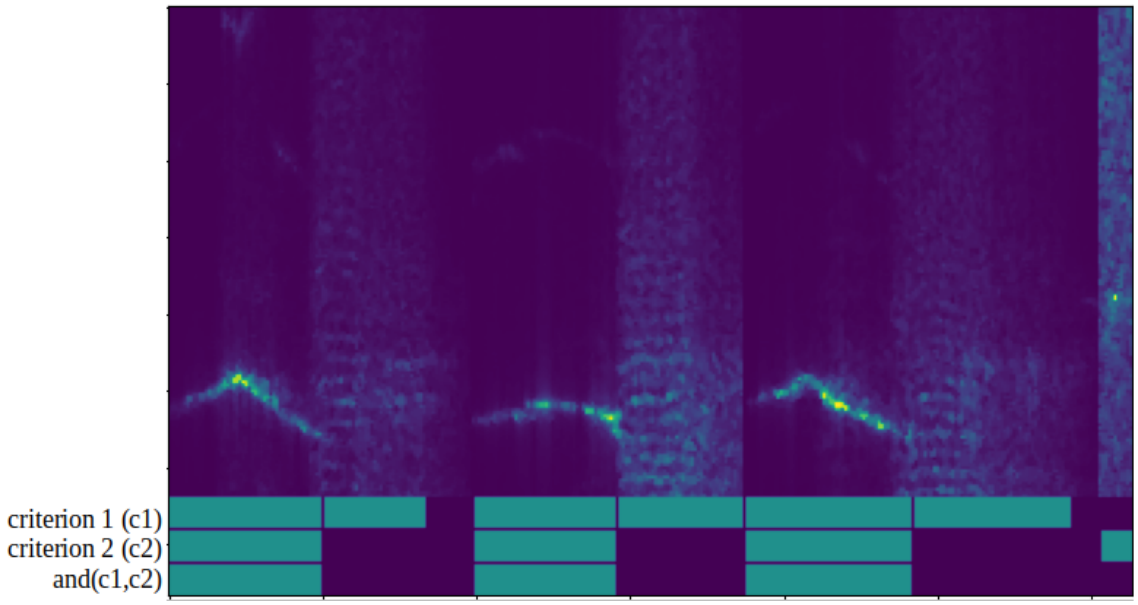


Figure 3.2: Examples of detection criteria: Demonstration of the twofold thresholding application. The green bars of the first two lines show the detected vocalizations by each criterion, whereas the third-line green bars are their intersection. Segments were spliced for purposes of visualization.

only time frames where the peak energy value P_i is larger than the mean spectral energy (M_i) of a 60kHz range around the frequency of the peak energy (truncated if the range goes below 30kHz or above 110kHz). If E_{ij} denotes the energy value at time frame i and frequency j, the equations describing the two quantities above, are the following:

$$P_i = \max_{j=30,\ldots,110kHz} E_{ij} \tag{3.4}$$

$$M_i = \frac{1}{N_f} \sum_{j=\max(p_i-30kHz,30kHz)}^{\min(p_i+30kHz,110kHz)} E_{ij} \tag{3.5}$$

where the step of j is equal to 0.5 kHz and, as a result, $N_f = 2 \cdot (\min(p_i + 30kHz, 110kHz) - \max(p_i - 30kHz, 30kHz))$, and p_i is the frequency of the peak energy at time frame i, i.e. $p_i = \arg\max_j(E_{ij})$.

Both criteria TT and FT are applied on each short-term frame i as follows: the threshold conditions require that the spectral energy is higher than 50 percent of the dynamic threshold computed in step 1 and that the maximum energy is larger than the mean spectral energy by a factor of 3.5. Let V be a sequence of frame-level vocalization decisions, i.e. $V_i = 1$ if time frame i is part of a vocalization and $V_i = 0$ if not. Then the above rule can be expressed as follows:

$$V_i = \begin{cases} 1 & \text{if } (S_i > t \cdot T_i) \text{ AND } (P_i > f \cdot M_i), \text{ where } t = 0.5 \text{ and } f = 3.5 \\ 0 & \text{else} \end{cases} \tag{3.6}$$

Both factors t and f have been selected after experimentation and can be considered as configurable (see Section 3.3).

After the twofold thresholding rule has been applied as described above, we apply a smoothing step. More specifically, after the thresholding the sequence V of 1s and 0s occurs,

this sequence is smoothed using a moving average filter with a duration of 20 ms, so that the neighborhood of the possible vocalization is taken into account:

$$F_i = \frac{\sum_{j=0}^{L-1} V_{i-j}}{L} \tag{3.7}$$

where L is the size of the filter. As a final step, if successive positive frames found in F are separated by <11 ms, they are concatenated to form segments of mice vocalizations. Vocalizations of duration <5 msec, are filtered out, since practical evaluation showed that in most cases these are false positive detections.

3.2 Online USV Detection

In addition to the offline detection that is standard for most USV analyses, we also developed an online version of AMVOC (i.e. using streaming sound recorded from the computer's soundcard). This is achieved by following the aforementioned analysis steps, though these cannot be applied to the whole signal at once, since, in a real-time setup, the signal is recorded simultaneously with the detection procedure. For this process, we wanted the processing interval to be as small as possible, in order to get the detected vocalizations fast. On the other hand, if we process the signal more often, the probability of cutting a vocalization in the border between two successive blocks is increased. Additionally, the signal statistics that need to be calculated for the detection steps described in the preceding section will become less robust if they are computed on smaller segments. We therefore chose to process the signal in blocks of a fixed duration, which for our case has been set equal to 750 ms. A 750ms window provides a long enough period for multi-syllable sequences of longer (>100ms) USVs to be captured, while not being so long of a window that feedback could not be provided quickly enough during a behavioral experiment.

The main algorithmic difference between the online and the offline detection is the calculation of the dynamic threshold. If we denote k as the current block, the dynamic threshold is the same for all frames belonging to that block and it is computed as follows:

$$T_k = 0.3\frac{\sum_{j=1}^{k} B_j}{k} + 0.7 \cdot B_k \tag{3.8}$$

where B is the total spectral energy block in the 30-110 KHz frequency range (as described in Section 3.1). In simple terms, the block's threshold is computed as the weighted average of the mean of the spectral energies of the blocks recorded up to that point and the current block's spectral energy. The exact weights (0.3, 0.7) were chosen after extensive experimentation. The sequence of thresholds that occurs is then multiplied by the threshold percentage t, exactly as in the offline method (Section 3.1). Another difference between the offline and the online methodologies is that for the online approach, we have selected to use an overlapping block. In particular, we always process the newest 750 ms recorded segment plus the last 100 ms of the previous block. In this way we add a minor computational delay (as we repeat the process for 13% of the data), and we manage to eliminate the errors caused by USVs being split between two successive blocks, and therefore lost by the detection method. The rest of the procedure is the same as the offline method and it is applied every 750 ms.

3.3 Vocalization Detection Configuration

As mentioned earlier, the two parameters which determine the vocalization detection procedure is the threshold percentage t and the factor f; this would mean that energy of a

60 kHz area M_i around the frequency of peak energy p_i must surpass peak energy P_i by a factor of f. The user can change these values according to the expected recording conditions and application requirements. Parameters used in our current study were optimized to include small events while minimizing false positives. To select these parameters, we used Dataset D1 (Figure 3.3). As expected, increasing either of these parameters results in a more strict thresholding, which means that temporal precision of detected increases, and temporal recall decreases. The opposite is observed when either of the parameters is reduced. From a more qualitative point of view, increasing the threshold might result in splitting a vocalization with relatively low peak energy in intermediate time frames. On the other hand, a very low threshold can merge separate vocalizations.

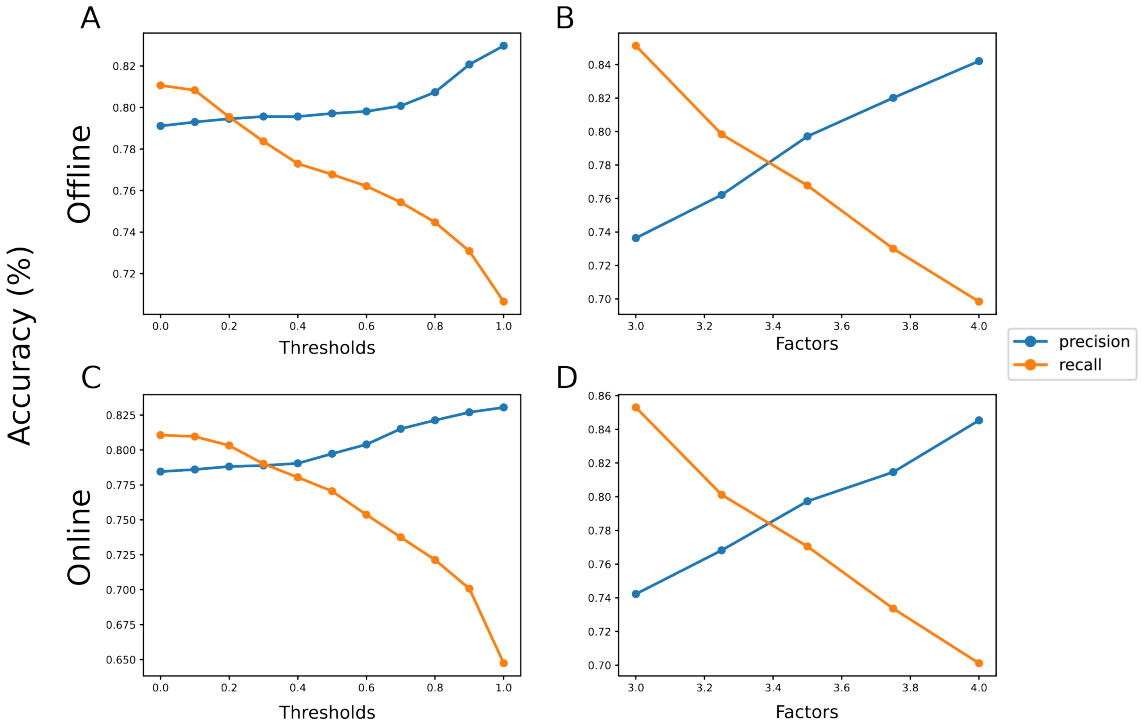


Figure 3.3: Effect of changing parameters on precision and recall. A and B) Changes in precision and recall during offline detection when thresholds t and factors f are changed. C and D) Changes in precision and recall during online detection when thresholds and factors are changed. Optimal threshold and factor were determined to be the same, 0.5 and 3.5, respectively, in both detection modes.

3.4 Experimental evaluation of the AMVOC detection method

Our objective was to design and implement a robust (in terms of detection performance) but also computationally efficient USV detection method, as our vision was to build a real-time and online pipeline. In fact, AMVOC's main value, compared to other methods, lies in the fact that it can also be used for online vocalization detection. We compared our proposed detection methodology with MSA (Mouse Song Analyzer-2 versions) (Arriaga et al. 2012; Chabout et al. 2015) and cite Peter DOI), MUPET (Van Segbroeck et al. 2017), VocalMat (Fonseca et al. 2021), and DeepSqueak (Coffey et al. 2019). Due to the vast range of possible parameters that could be tuned in each method, we used default settings, unless there were other documented settings that were used (Chabout et al. 2017). In order to evaluate and compare the aforementioned methods, we used Dataset D1, which consists of 9 audio segments of 5-10 seconds each, containing 245 annotated syllables in total (see 1.5).

To evaluate the range of experimental contexts recordings could be taken from, we split the recordings into two categories: normal and noisy. Normal parts of the recording are the ones where the vocalization detection is relatively straightforward, because the energy easily surpasses the background energy. Noisy parts contain background noise, which makes the detection more difficult and ambiguous, even for the human eye observing the raw spectrograms. The evaluation metrics are calculated separately for the two categories.

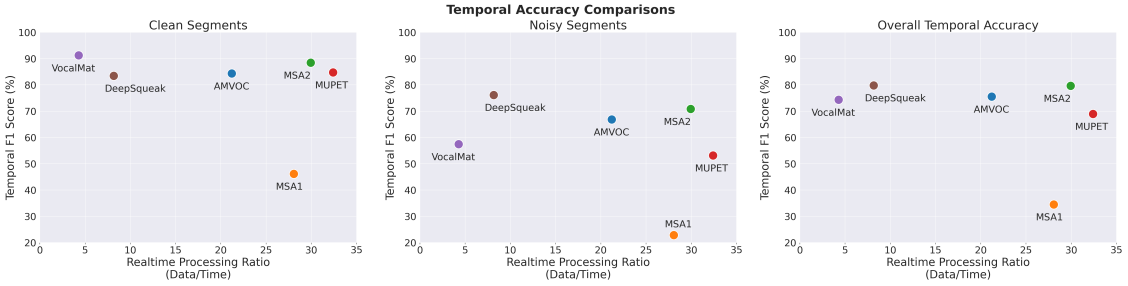
For evaluation and comparison we adopted two performance metrics, the temporal F1 and the event F1 score (see Section 2.8):

AMVOC outperforms the other methods with respect to event F1 score, both in clean and noisy segments of the recordings, whereas MSA2 and DeepSqueak performed slightly better than the others with respect to temporal F1 score (Table 3.1), largely due to the more successful detection in the noisy parts of the recordings.

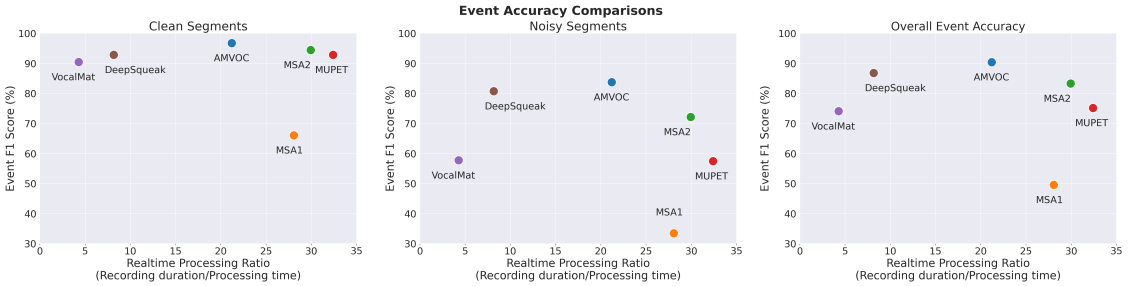
		AMVOC						
F1 score		offline	online	MSA1	MSA2	MUPET	VocalMat	DeepSqueak
Temporal	Normal	84	**85**	46	88	85	91	83
	Noisy	67	68	23	71	53	57	**76**
	Average	75.5	76.5	34.5	**79.5**	69	74	**79.5**
Event	Normal	**97**	**97**	66	94	93	90	93
	Noisy	**84**	83	33	72	57	58	81
	Average	**90.5**	90	49.5	83	75	74	87

Table 3.1: F1 scores of our proposed method and other methods.

Using Dataset D1, we also assess the trade-off between processing speed and detection (Table 3.2).



(a) Temporal F1 score vs Realtime Processing Ratio.



(b) Event F1 score vs Realtime Processing Ratio.

Figure 3.4: Accuracy of AMVOC and Other Methods A-B) Event and temporal F1 score vs Realtime Processing Ratio of different USV detection methods compared against our ground truth data in different qualities of recordings.

Methods	Real-time Processing Ratio
MUPET	**32.4**
MSA2	29.9
MSA1	28.1
AMVOC	21.2
DeepSqueak	8.2
VocalMat	4.3

Table 3.2: Real-time Processing Ratio of all compared methods. Real-time processing ratio is defined as $r = \frac{d}{p}$, where d is the duration of the recording and p its processing time and is shown for each method. The processing time is calculated as the time needed to just detect the USVs. The experiments carried out to compute the real-time ratio were executed for 5 different recordings, 3 times for each, and the average time for each method was calculated. Obviously, a high real-time processing ratio means that a small processing time is required in order to detect the vocalizations of a certain signal (e.g. $r = 30$ means that the respective method is 30 times faster than real-time, meaning it takes 1 minute to process 30 minutes of audio information).

AMVOC had an intermediate real-time processing ratio to detect the vocalizations (Table 3.2). MUPET was the fastest method, whereas VocalMat and DeepSqueak were the slowest (Table 3.2). The reason for the latter two methods being slower is likely due to their image processing steps used to detect USVs. It is also meaningful that we take into account both of the two aforementioned metrics, since it may be important for particular experimental requirements that a certain method combines accurate and fast detection. We compared the average F1 score with the real-time Processing Ratio for both temporal and event F1 scores (Figure 3.4a and 3.4b). AMVOC, DeepSqueak and MSA2 achieved the highest temporal and event F1 score, but AMVOC had a considerably better time performance relative to DeepSqueak and a better event F1 score in noisy segments compared to MSA2.

Chapter 4

Deep unsupervised learning for mouse vocalization clustering

The goal of this chapter is to present a method for unsupervised clustering of the detected vocalizations, by using a convolutional autoencoder, from which features are derived. Our unsupervised approach, both for feature extraction and, of course, for the clustering, is chosen bearing in mind that we wanted to explore new, possibly meaningful groupings of USVs, without any limitations imposed by humans. In this Section we describe the procedure of building and training the autoencoder.

4.1 Unsupervised learning pipeline

The procedure consists of all the steps described in Section 2.2, adjusted to our specific problem. To better clarify and organize the procedure we followed, we will parallelize it to the system design pipeline. The first stage refers to the sensor, with in our case is he microphone since it provides us with the recorded signal. This signal can then be used and processed for feature generation/extraction.

4.1.1 Feature generation

For this stage, we don't use the raw signal, but the spectrogram calculated from the signal, as we have already mentioned. The spectrogram is necessary for the vocalization detection (see Chapter 3). After the vocalization detection, we have a set of spectrogram segments (images), each one corresponding to a syllable and our goal is to use these images to extract features that can best both discriminate and group vocalizations. In order to achieve this, we use a convolutional autoencoder (see Section 2.5.4.2).

4.1.1.1 Proposed autoencoder architecture and training

To train the autoencoder, we used Dataset D2 (see Section 1.5) and we calculated the spectrogram for each of its recordings. This dataset contains 22,409 detected syllables. Each vocalization is represented by a spectrogram in the detected time interval and defined frequency range. Therefore, these spectrograms vary in width, which corresponds to the respective syllable duration. So, we have to specify the width of the images that we are going to feed to the autoencoder, since the frequency y axis is the same for all spectrograms: 80 kHz range (from 30-110 kHz) and 0.5 kHz frequency resolution, resulting in a dimension of 160. Selecting a fix sized time dimension for our spectrograms requires taking into consideration a tradeoff between losing important information from the larger spectrograms (if we crop them) and reducing the importance of the shape and details of the smaller

spectrograms (if we zero-pad them), which are more numerous than large spectrograms (Figure 4.1).

In order to decide the final fix size of the spectrograms to feed the autoencoder, we plotted a histogram of the initial durations of all the detected syllables in the training set (Figure 4.1). To ensure uniform sizing, we zero padded small spectrograms, and cropped larger spectrograms keeping the central part of the image. Based on the histogram we selected a fix-sized duration of 64 windows since it is larger than both the mean and the median of the durations (Figure 4.1). Using this size, we noted that it balanced the information tradeoff mentioned above. It is also a power of 2, which is convenient for the pooling operations in the autoencoder. This length of 64 frames corresponds to 64 time frames x 0.002 sec/time frame = 0.128 sec = 128 ms. The aforementioned process of cropping or expanding spectrograms to a fix-sized width of 64 windows leads to spectrograms of a final resolution of 64 time frames x 160 frequency bins.

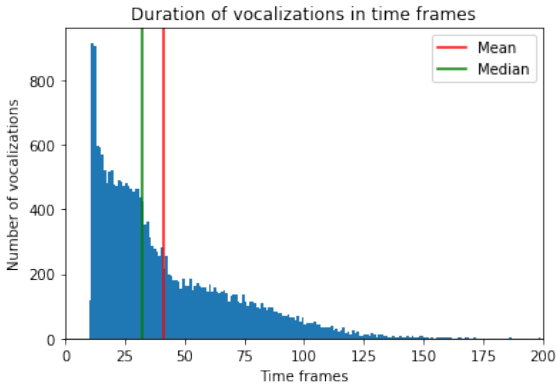


Figure 4.1: Histogram of the duration of the vocalizations in time frames Each time frame corresponds to a 2 ms duration.

The training set consists of 22,409 images, each one with size 64x160. We fed the images to the encoder, which is a convolutional neural network with 3 convolutional layers, each followed by a max pooling layer (Figure 4.2). The first convolutional layer uses 64 filters, with dimensions 3x3 each.

After that, a max pooling layer decreases the spatial dimensions of the images by a factor of 2. This means that the output of the max pooling layer is a 32x80x64 representation. The next convolutional layer consists of 32 filters, with a max pooling layer generating an output with dimensions 16x40x32. The third and final layer includes 8 filters, and a max pooling layer, resulting in a convolutional activation map for each image with dimensions 8x20x8. This flattened intermediate representation is the feature vector that uniquely describes the input image (i.e. the code).

In order for the convolutional encoder to be trained, and because the task is unsupervised, the second part of the autoencoder (the decoder) is responsible for reconstructing the image we fed to the encoder from the intermediate representation (Figure 4.2). The decoder reverts the steps of the encoder, using 32, 64 and 1 filter in the last layer. In each decoder layer, we use filters of size 2x2 and a stride of 2, so that after each layer the size of the representation increases by 2 and the final output of the autoencoder is an image with the same size as the original input. After each convolutional layer, the ReLU activation function (see Section 2.5.2.2) is used, since we want the activation maps to consist of positive values, except from the last deconvolution of the decoder, where the Sigmoid activation function is necessary for the reconstruction of the image and the calculation of the Binary Cross Entropy Loss function, as it outputs a value between zero and one.

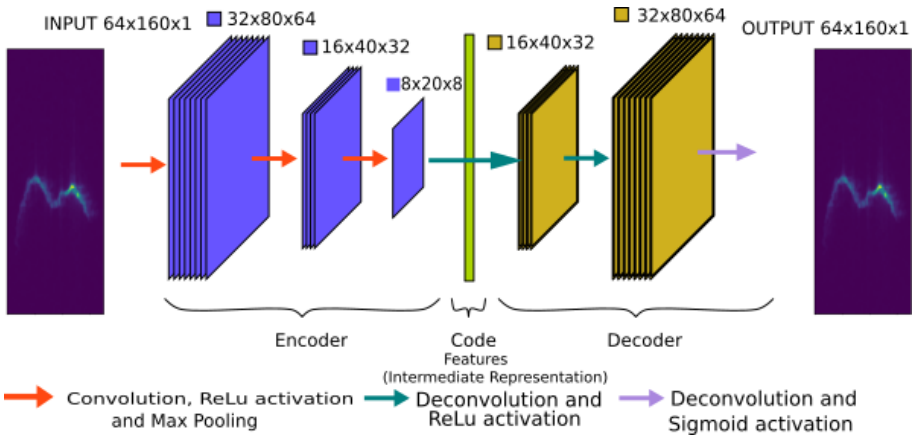


Figure 4.2: AMVOC convolutional autoencoder: Architecture used for the autoencoder in AMVOC.

Parameter tuning As described above, the basic parameters for configuring our proposed autoencoder procedure are the following:

- *Number of layers of the encoder.* We tested a range of different numbers of layers (2-5 layers). Using 2 layers was too few to effectively learn the complex structure of spectrograms. More than 4 layers had too many parameters, which slowed down the training, resulting in a bigger loss, and required more epochs to be trained, while the final reconstruction had no considerable differences compared to the one from the 3-layer-encoder.

- *Number of filters per layer* We used the most filters in the first layers (as is typical for classical neural networks), and reduced the number of filters as we went deeper in the network. We tested the autoencoder with varying numbers of layers. Fewer filters resulted in losing information from the images, while more resulted in too many parameters to be trained. The critical choice we had to make was the number of filters in the last encoder layer (that also define the size of the code, since the size of 3rd dimension is equal to the number of filters), the output of which we use as the representation for the specific image in the clustering task. We tested 2, 4 and 8 filters. Using 8 filters resulted in smaller loss, as expected, although using 4 filters were enough for the reconstruction of the images, meaning enough features for the reconstruction were extracted with fewer filters. We selected 8 filters in our final design in order to ensure that all the details of the various shapes of the vocalizations are properly extracted. (see Figure 4.3)

- *Filter size* We experimented with 3x3, 5x5 and 3x5 kernels. A 3x5 kernel appeared reasonable because of the non-square shape of the images, but a 3x3 kernel gave us the best results.

- *Size of max pooling kernels* We experimented with reducing the image size by 2 in both dimensions, or by 2 in time dimension and by 4 in frequency dimension due to the non-square image. A 2x symmetrical reduction provided better results.

As far as other hyperparameters are concerned, we used the Adam optimizer, with learning rate equal to 0.001 and batch size equal to 32.

Training epochs were determined experimentally. In fact, 2 or 3 epochs was enough for a good reconstruction of the images, and loss did not decrease much after 3 epochs (Figure

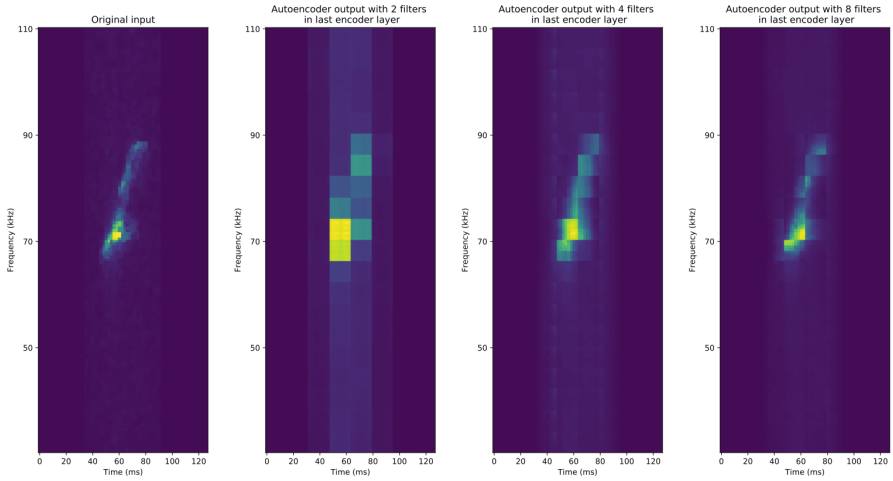


Figure 4.3: Examples of image reconstruction with AMVOC's autoencoder after training, using 2, 4 and 8 output filters. Data is extracted from the input image (left) and used to reconstruct the three images (right).

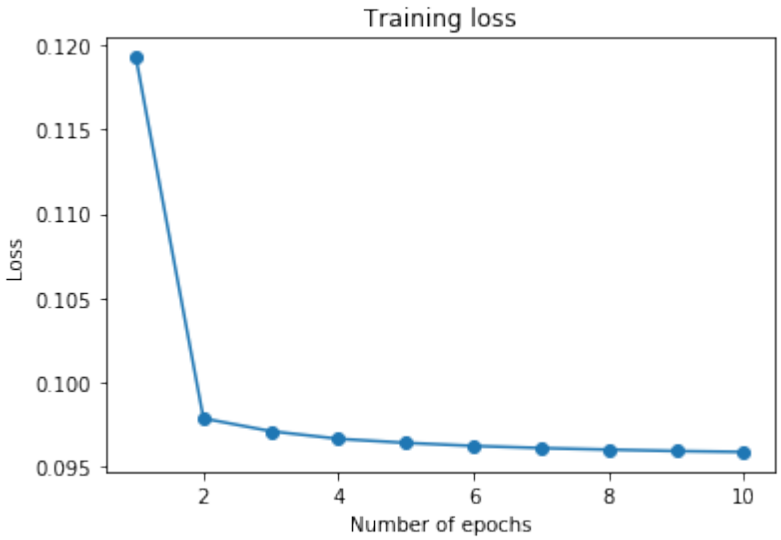


Figure 4.4: Effect of the number of training epochs on measured training loss.

4.4). We also did not want to overfit to the training data. Thus, we elected to train the model for just 2 epochs.

An example of the input and output of the autoencoder is shown in Figure 4.3. The input comes from a recording that was not used in the training Dataset D2. The reconstruction is lossy, likely due to our use of an undercomplete autoencoder.

After the model has been trained in the unsupervised manner described above, it is ready to be used in the feature extraction procedure. An audio file is selected and converted to a spectrogram, and individual USVs are detected. The raw spectrograms of the USVs are fed to the autoencoder in batches of 32 and the intermediate representations are derived. These are the feature vectors. Each flattened feature vector has a dimension of 1,280 (8x20x8) after a dimensionality reduction from a dimension of 10,240 in the initial flattened vector, since each image started with a shape of 64x160.

4.1.2 Feature selection

The next stage of the pipeline is feature selection. We use a Variance Thresholder (see Section 2.6.1.1) to exclude features with the smallest variance.

After extensive qualitative experimentation, and supposing we have M N-dimensional feature vectors $\mathbf{x}_i$, $i = 1, ..., M$ we selected a threshold equal to:

$$v_t = 1.2 \cdot \frac{1}{N} \sum_{j=1}^{N} \sigma_j^2 \tag{4.1}$$

where σ_j^2 is defined in Equations 2.63 and 2.64 and refers to the variance of feature j. So, v_t is actually the mean of the features' variances; features with variance less than v_t will be excluded, which results in a dimensionality reduction of a factor approximately equal to 4.

4.1.3 Feature pre-processing

Before we use the features, we use some pre-processing steps.

4.1.3.1 Feature scaling

First, as far as feature scaling is concerned, we chose to standardize them using a Standard Scaler (see Section 2.6.2.2), in order to make their values comparable and succesfully handle outliers.

4.1.3.2 Dimensionality reduction

Next, we used PCA to further reduce the dimensionality of the feature vectors. Since we do not know the number of components beforehand, we choose the smallest number of components which maintains 95% of the variance of the features before the PCA. Overall, our goal was to both extract many features from the images using the encoder, so that details of the images are taken into account, and simultaneously reduce them as much as possible by ignoring the non-significant features.

The feature extraction pipeline we described uses a deep architecture which learns the most significant features. However, features can also be extracted from the spectrogram "manually". We are now going to present a traditional feature extraction procedure, resulting in so-called *hand-crafted* features.

4.1.4 Baseline feature extraction

To evaluate the quality of AMVOC's deep feature extraction and clustering, we wanted to compare clustering on deep features against clusters derived from hand-crafted acoustic parameters. The hand-crafted features were measured as follows:

1. We first calculate the spectrogram in the specific time segment that corresponds to the vocalization and in the defined frequency range (30-110 kHz).

2. We then perform frequency contour detection:

    - Detect the position and the value of the peak energy in each time frame. If we denote the spectrogram value at time frame i and frequency j as E_{ij}, the equations describing the two quantities above, are the following:

$$p_i = \underset{j=30,...,110kHz}{\arg\max} E_{ij} \tag{4.2}$$

$$P_i = \max_{j=30,\ldots,110kHz} E_{ij} \tag{4.3}$$

- Use a thresholding condition to keep only the points i where the peak energy is higher than 20 percent of the highest energy value in the specific time interval:

$$\text{if } P_i > t \cdot \max_i P_i, \text{ where } t = 0.2, \text{ keep point } (i, p_i) \tag{4.4}$$

- Train a regression SVM to map time coordinates to frequency values, using the chosen points (i, p_i) as training data.
- Predict the frequencies for the same time range. After that, for each vocalization, a frequency contour c is created, along with a corresponding time vector v, which matches every frame i to its actual time of occurrence. This estimated c sequence captures the "most dominant" frequency in each time frame, so we can think of it as a spectral shape sequence of each mice vocalization.

3. After the frequency contour c is produced, we proceed to the feature extraction step. We selected 4 different features, all based on the frequency contour:

- Duration of the vocalization d. If we denote the number of frames of which the vocalization consists as N:

$$f_1 = d = v[N-1] - v[0] \tag{4.5}$$

- Time position of the minimum frequency (of the predicted frequencies) (f_2'), normalized by the duration of the vocalization:

$$f_2 = \frac{f_2'}{d} = \frac{v[\arg\min_{i=0,\ldots,N-1} c_i] - v[0]}{d} \tag{4.6}$$

- Time position of the maximum frequency (of the predicted frequencies) (f_3'), normalized by the duration of the vocalization:

$$f_3 = \frac{f_3'}{d} = \frac{v[\arg\max_{i=0,\ldots,N-1} c_i] - v[0]}{d} \tag{4.7}$$

- Bandwidth is calculated as the difference between the first and the last predicted frequency value (f_4'), normalized by the mean frequency of the vocalization m_f:

$$f_4 = \frac{f_4'}{m_f} = \frac{c_{N-1} - c_0}{m_f}, \text{ where } m_f = \frac{\sum_{i=0}^{N-1}(c_i)}{N} \tag{4.8}$$

If we interpret the frequency contour as a 2-dimensional graph, where the x-axis corresponds to time and the y-axis to frequency, the 2nd feature is the x-position of the minimum of the curve, the 3rd feature is the x-position of the maximum of the curve and the 4th feature is the normalized difference between y-positions of first and last point (see Figure 4.5). For example, the last feature can discriminate contours with different slopes. After the features are extracted, we scale them using a standard scaler (see Section 2.6.2.2).

4.1.5 Clustering

The next step of the system design pipeline is the Classifier design. Since we deal with an unsupervised learning problem, this stage refers to training a clustering algorithm using the extracted features, regardless if they are hand-crafted or extracted using the deep learning pipeline.

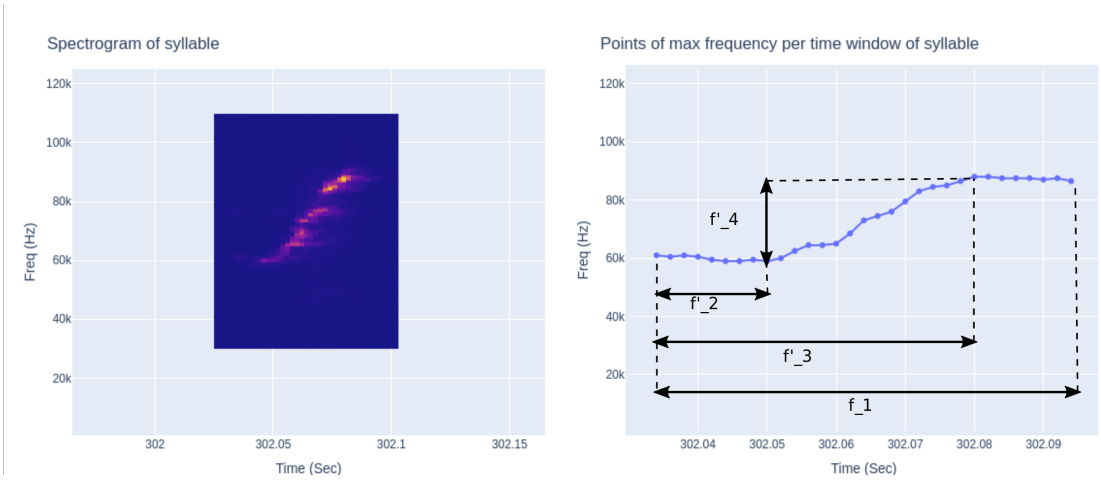


Figure 4.5: A syllable (left) and its frequency contour (right), used for the extraction of the hand-crafted features.

Each cluster should consist of vocalizations that share some common features that allow them to belong in the same group. Since we wanted the user to be able to choose the number of clusters, we chose clustering algorithms where we can predefine this parameter. We selected testing the following clustering methods: Agglomerative, Gaussian Mixture Models, K-Means, Mini-Batch K-Means and Birch (often used as alternative to Mini-Batch K-Means).

For this task to be visualized and to give the users the opportunity not only to select the clustering settings, but also to inspect the resulting clusters and the syllables assigned to each cluster, we have implemented a user-friendly GUI, where the user can choose one of these different clustering methods and the number of clusters, in a range from 2 to 10.

Figure 4.6 presents the whole procedure described in the last sections, regarding the deep feature extraction approach.

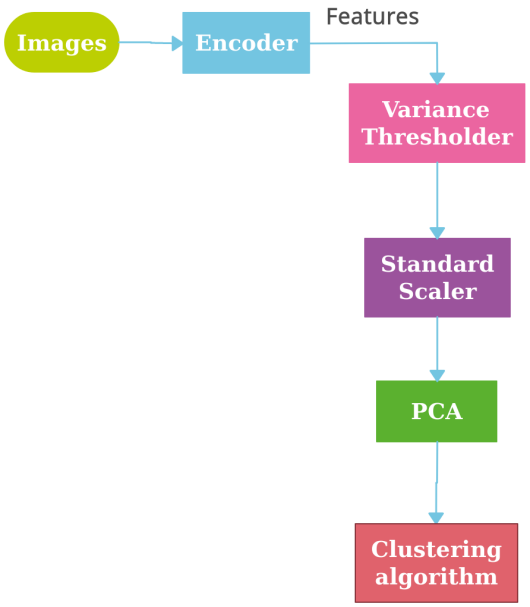


Figure 4.6: Overview of deep feature extraction procedure: Flow diagram of the general procedures used to take image data from USV spectrograms into clusters.

Figure 4.7 presents an example of the produced clustering of syllables using the K-Means algorithm and 6 clusters. For 2D visualization purposes, we have used the t-SNE dimensionality reduction method.

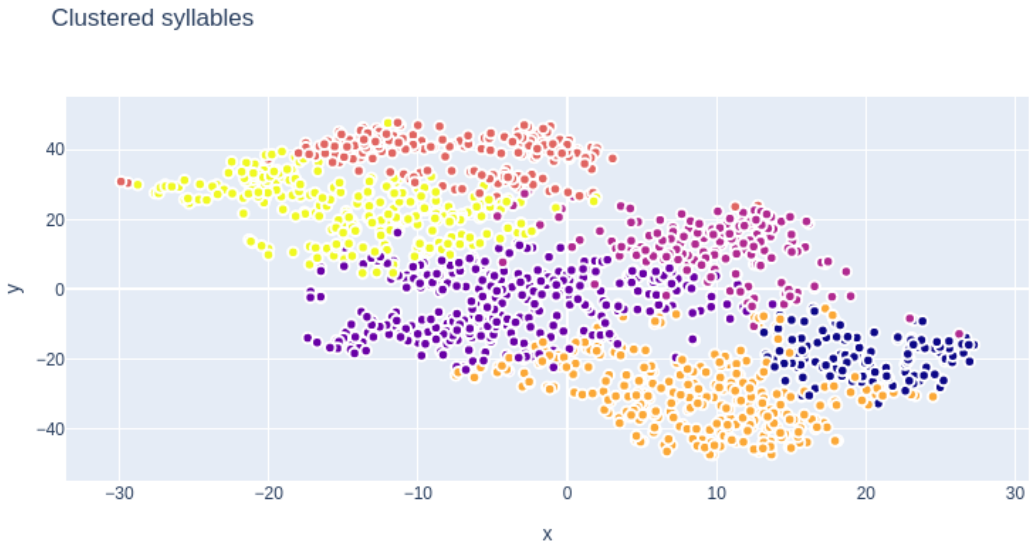


Figure 4.7: Cluster example using deep features with K-Means clustering and 6 clusters. Each point corresponds to a syllable.

4.1.6 Experimental evaluation of the AMVOC clustering method

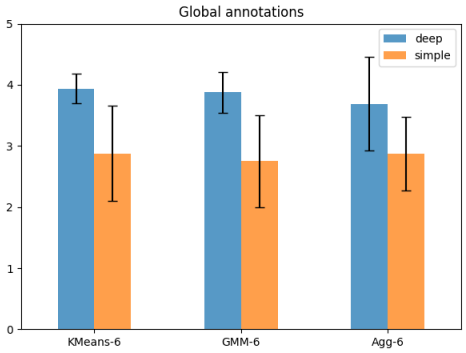
The last pipeline stage is the evaluation of the model we have trained (in our case, the evaluation of the resulting clustering.) Our primary goal is to compare the deep feature extraction method (see 4) to the baseline feature extraction method (using hand-crafted features described in Section 4.1.4) by evaluating the derived clustering of the two kinds of features. We also want to evaluate how well the different clustering methods perform in grouping vocalizations with common features. To achieve these two goals, we used data from four different annotators, two of whom are domain experts, and the remaining two are not. By using the AMVOC GUI, each annotator evaluated the 4 recordings from Dataset D3. Specifically, the annotators evaluated 3 different clustering setups for each recording:

1. Agglomerative clustering with 6 clusters
2. Gaussian Mixture clustering with 6 clusters
3. K-Means clustering with 6 clusters

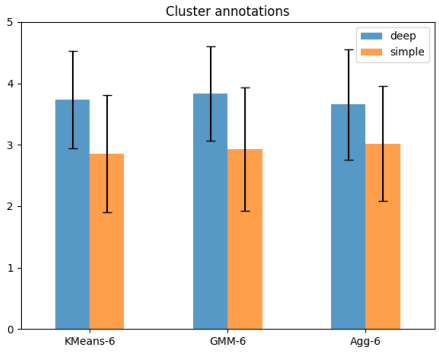
In order to ensure impartiality and objectivity, the annotators evaluated the clustering derived from both feature extraction methods (named as Method 1 for deep features and Method 2 for hand-crafted features), without prior knowledge of which Method refers to the deep features or the hand-crafted. Using a scale from 1 to 5, 5 being the best, the evaluation metrics used are the following:

1. Global annotations: The annotator defines a score to describe how successful the whole clustering is.
2. Cluster annotations: The annotator defines a score to describe how successful each cluster is.

3. Point annotations: The annotator selects points from different clusters and declares whether they should be approved or rejected in the specific cluster to which they have been assigned. Approximately 100 points were annotated by each user per setup, for each method.



(a) Global annotation evaluations: Mean and standard deviation of global clustering scores for each clustering setup.



(b) Cluster annotation evaluations: Mean and standard deviation of cluster annotations for each clustering setup.

Figure 4.8: Global and cluster annotation scores.

First we evaluated the global annotations (Figure 4.8a). For each annotator i, we calculated the mean score μ of each setup s (KMeans-6, GMM-6, Agg-6), using the scores from the 4 recordings:

$$\mu_{i,s} = \frac{1}{4} \sum_{j=1}^{j=4} G_{i,s,j} \tag{4.9}$$

where counter j refers to the 4 recordings and $G_{i,s,j}$ is the global score of setup s, in recording j, set by annotator i. Then, the mean m and standard deviation d of these mean scores of the 4 users is calculated (Figure 4.8a).

$$m_s = \frac{1}{N_a} \sum_{i=1}^{i=N_a} \mu_{i,s} \tag{4.10}$$

$$d_s = \sqrt{\frac{1}{N_a} \sum_{i=1}^{i=N_a} (\mu_{i,s} - m_s)^2} \tag{4.11}$$

where Na is the number of annotators (in our case, 4). Based on the annotation results, deep feature extraction yields better clustering results than simple feature extraction, with all setups (Figure 4.8a). The setup does not seem to affect the performance of the clustering very much, as far as the mean values are concerned. However, we note that K-Mmeans and GMM deep feature extraction scores have a smaller standard deviation than Agglomerative scores (S.D. values, K-Means = 0.24, GMM = 0.33, and Agglomerative = 0.76), and compared to their respective simple feature extraction scores (S.D. values, K-Means = 0.78, GMM = 0.75, and Agglomerative = 0.60). A lower standard deviation means that the annotators mostly agreed at their scores.

Next we looked at the cluster annotations scores (Figure 4.8b). For each setup and annotator, we calculated the mean scores μ' of all the 6 clusters in the 4 recordings:

Global annotation Scores	t value	p value
K-Means-6	5.0	$1.7{\cdot}10^{-4}$
GMM-6	5.1	$1.3{\cdot}10^{-4}$
Agglomerative-6	3.1	$7.2{\cdot}10^{-3}$

Table 4.1: Statistical analysis for the global annotation scores. We have performed a paired t-test to infer the statistical significance of the differences between deep and simple features.

Cluster annotation Scores	t value	p value
K-Means-6	6.7	$1.3{\cdot}10^{-9}$
GMM-6	6.5	$3.4{\cdot}10^{-9}$
Agglomerative-6	5.3	$8.3{\cdot}10^{-7}$

Table 4.2: Statistical analysis for the cluster annotation scores. We have performed a paired t-test to infer the statistical significance of the differences between deep and simple features.

$$\mu'_{i,s} = \frac{1}{4 \cdot c_s} \sum_{j=1}^{j=4} \sum_{k=1}^{k=c_s} C_{i,s,j,k} \tag{4.12}$$

where counter j refers to the 4 recordings, k to the number of clusters of each setup c_s (in our case, 6 for all setups) and $C_{i,s,j}$ is the cluster specific score of cluster k of setup s, in recording j, set by annotator i.

These mean scores were then used to calculate the mean and standard deviation values for each setup:

$$m'_s = \frac{1}{N_a} \sum_{i=1}^{i=N_a} \mu'_{i,s} \tag{4.13}$$

$$d'_s = \sqrt{\frac{1}{N_a} \sum_{i=1}^{i=N_a} (\mu'_{i,s} - m'_s)^2} \tag{4.14}$$

where N_a is the number of annotators (in our case, 4). The results of the cluster-level annotation scores are consistent with the ones from the global annotation scores (Figure 4.8a and 4.8b), except for the standard deviation observation. It should be noted that for cluster annotation scores, the standard deviation metric does not provide much information about the different feature extraction methods or the different setups.

In order to draw safe conclusions, we ran a paired t test between annotation scores for the two features types (see Tables 4.1, 4.2). As it is stated in Tables 4.1 and 4.2, both global and cluster annotation scores have a high t value and a very low p value, indicating a strong statistical significance of the difference between the scores for deep and simple features.

We next assessed the percentage of vocalizations each user approved for each cluster based on their unsupervised assignment for both deep and simple methods (Figure 4.9). If we denote the approved vocalizations of user i as a_i, and the total vocalization annotations they have made as t_i, then the mean percentage M of approved vocalizations is calculated as:

$$M = \frac{1}{N_a} \sum_{i=1}^{N_a} \frac{a_i}{t_i} \tag{4.15}$$

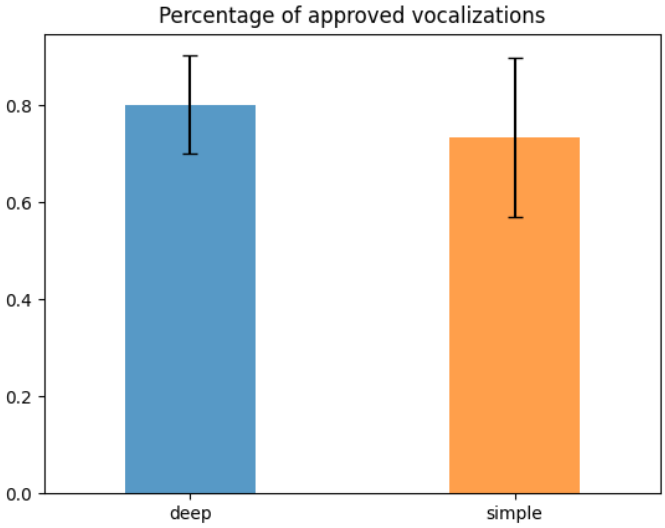


Figure 4.9: Mean percentage of approved vocalizations for point annotation evaluation.

It is clear that users more frequently approved vocalizations clustered by the deep feature extraction method. The standard deviation of the average approval rate that referred to this method is also smaller than for hand-crafted features (S.D values, Deep = 0.10 and Hand-crafted = 0.16), indicating a more confident average value.

Overall, the deep feature extraction method outperforms the simple method in all terms:

- Global clustering evaluation is 37% higher for the deep approach (Figure 4.8a)
- Cluster-specific evaluations are also 30% higher on average for the deep approach(Figure 4.8b)
- Average point-level evaluation was 10% higher for the deep approach (Figure 4.9)

This suggests that the encoder has managed to indeed retrieve useful information of each image, resulting in feature vectors that enable a better clustering of the vocalizations. Further, this indicates that there are much more complex similarities and differences between vocalizations carried by the representations extracted from the encoder than what is available from hand-crafted features.

Chapter 5

Semi-supervised learning for refining mice vocalizations clustering

Up to this point, we have presented a completely unsupervised alternative to traditional feature extraction procedures for clustering of mice vocalizations, with a significant performance improvement as stated in Section 4.1.6.

Our goal is to further improve the clustering performance by introducing human intervention to the training of the autoencoder, by supplying information about the vocalizations in the form of pairwise constraints. This information will then be integrated in the loss function used for training the model.

This intervention is recording-specific, in the sense that after the clustering of vocalizations of a certain recording has been produced, the user of AMVOC tool has the chance to explore the possibility of improving the clustering by examining pairs of vocalizations and stating whether or not they should belong to the same cluster. So, we have a pre-trained model, which is retrained for the specific recording.

The whole process is basically the same as in *Semi-Supervised Deep Embedded Clustering* (see Section 2.7.1.5), with adaptations to match our application.

The ultimate goal is to be able to improve the clustering and use each feature vector along with its cluster label as *ground-truth* data to train a classifier (supervised task). This classifier can then be employed to predict the class of new vocalizations, possibly in online mode, and thus provide the users with information about the class of every detected syllable in real time.

A use-case scenario (where the user is e.g. a biologist who wants to explore new possible meaningful groupings) is presented in Figure 5.1.

5.1 Semi-Supervised Deep Embedded Clustering

Semi-supervised Deep Embedded Clustering is thoroughly described and explained in Section 2.7.1.4. Its primary goal is to further train an already pre-trained autoencoder, paying attention to clustering purity and human-defined pairwise constraints. The first is accomplished by integrating a KL Divergence term in the loss function, and the second by adding a term which penalizes close representations when they are connected with a cannot-link or distant representations when they are connected with a must-link.

5.1.1 Parameter initialization

The first step of DEC (see Section 2.7.1.4) is the training of the autoencoder and initializing cluster centroids μ_j by clustering feature vectors of the output of the encoder.

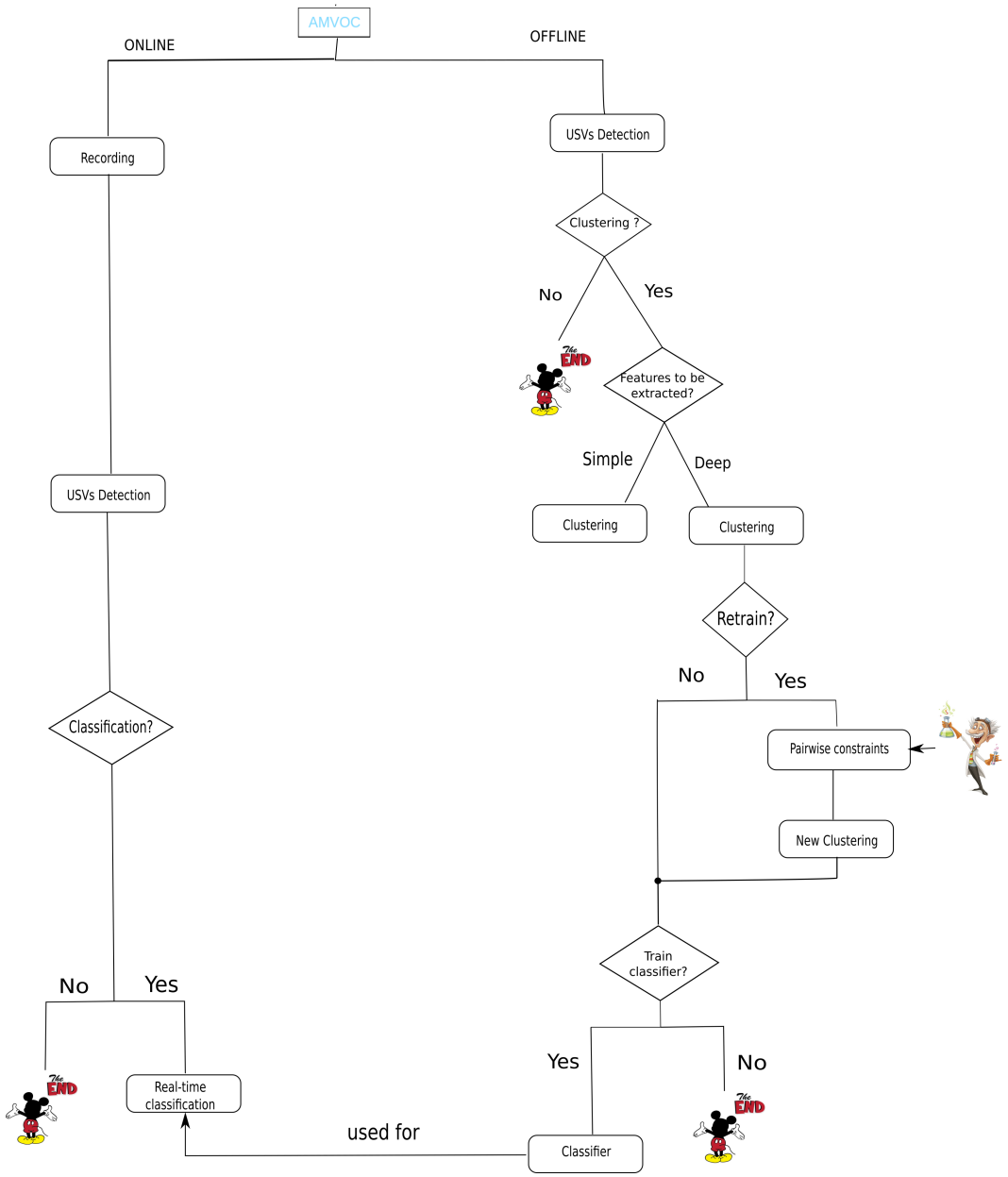


Figure 5.1: Use-case scenario.

In our case, we have trained the autoencoder as described in Section 4.1.1.1 and we can then proceed to a K-Means clustering with k clusters to derive the cluster centers μ_j.

5.1.2 Clustering with KL Divergence

To ensure a more clustering-oriented training of the autoencoder, we insert a KL Divergence term to the loss function, whose goal is to reduce the difference between our patterns distribution q_i (calculated using the cluster centroids μ_j) and a target distribution p_i (both defined in Section 2.7.1.4). This term is based on the clusters calculated using the

feature vectors at the output of the encoder and its aim is to emphasize training samples that were assigned to a cluster with high confidence.

However, as stated in Sections 4.1.1.1, 4.1.2, 4.1.3, the feature vectors dimension at the output of the encoder in our application is equal to 1280, which is reduced after applying feature selection and dimensionality reduction methods, meaning that the final clustering we use is done on the processed features.

Because of that, the clusters computed during the algorithm aren't finally used in our case. However, we decided to keep this approach and integrate KL Divergence to the loss function, since it can be useful for improving the representations for a clustering-oriented task.

5.1.3 Reconstruction Loss

Another difference to Ren et al. (2018) is that we also integrated the classical BCE Loss term in the loss function, like in typical autoencoder training, since we don't want the model to "forget" the reconstruction features and give very clustering or manually annotations oriented results. In this way, we hope to make the model more robust to, perhaps, false or confusing pairwise constraints.

5.1.4 Pairwise constraints

Pairwise constraints are defined by the user simultaneously with the first epoch of training, making this procedure an *active learning* task. More specifically, training is done in batches of 32 vocalizations; a predefined number of pairs per batch is selected and the two syllables are presented to the user, who declares whether they should or should not belong to the same cluster. In this way, pairwise constraints are set.

An important issue we dealt with was how to choose the pairs (let's say we annotate N pairs per batch). We didn't want it to be fully deterministic, but we also wanted to experiment with a non-totally-random choice. So, after some experimentations, we decided to sort the syllables i according to $\max_j q_{ij}$ (which expresses their most likely cluster, see Section 2.7.1.4), and therefore choose the N first syllables which belong to their likeliest cluster with the least confidence (have the smallest $\max_j q_{ij}$), compared to the other syllables of the certain batch. The second pattern of each pair is randomly selected.

We also experimented with the choice of a_{ij} (values of pairwise constraints matrix A). More specifically, we noticed that in general, due to the loss function type which involves the term $\left\|z_i - z_j\right\|$ (see Equation 2.103), the gradient of the loss function and concesquently the whole training is more affected by distanced representations, regardless of whether they are labeled as must-link or cannot-link; so, we wanted to ensure a more fair participation of pairs in loss definition. To this end, we didn't set $a_{ij} = 1$ or -1 as proposed in SDEC, but we wanted to use a descending function of distance $\left\|z_i - z_j\right\|$:

$$g(x) = \text{erf}\left(\frac{c}{x}\right) \tag{5.1}$$

where, in our case, we set $x = \left\|z_i - z_j\right\|$ and $c = +/-25$ (after experimenting) and erf is the so-called error function (see Figure 5.2). So:

$$a_{ij} = \begin{cases} \text{erf}\left(\frac{c}{\left\|z_i - z_j\right\|}\right), & \text{if } i-j \text{ must-link} \\ \text{erf}\left(\frac{c}{\left\|z_i - z_j\right\|}\right), & \text{if } i-j \text{ cannot-link} \\ 0, & \text{if we have no information about} i-j \end{cases} \tag{5.2}$$

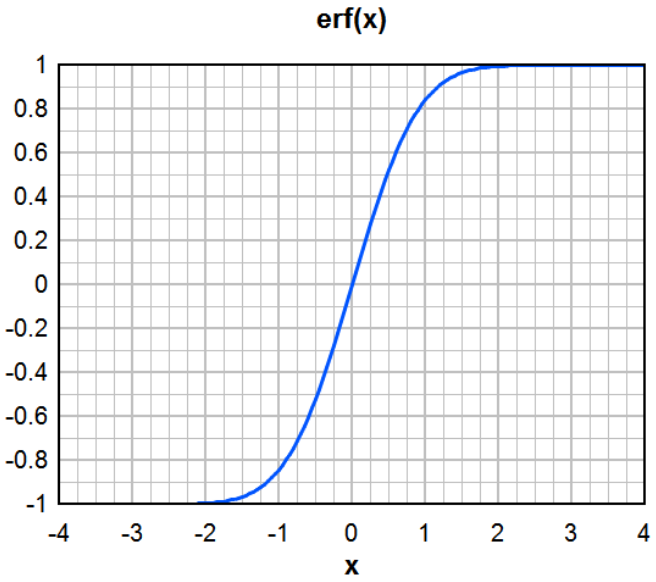


Figure 5.2: Error function.

5.1.5 Training

After these alterations, we define the loss function as:

$$L = \gamma_1 \cdot \text{BCELoss}(r, \hat{r}) + \gamma_2 \cdot KL(P\|Q) + \gamma_3 \cdot \frac{1}{N \cdot n_e} \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} \|z_i - z_j\|^2 \tag{5.3}$$

where n_e is the current number of epoch (added to reduce the impact of this term during further epochs, since the representations tend to have an increased magnitude as epochs proceed). The weights γ_1, γ_2 and γ_3 are configurable parameters and are set by the user, depending on which term they want to emphasize.

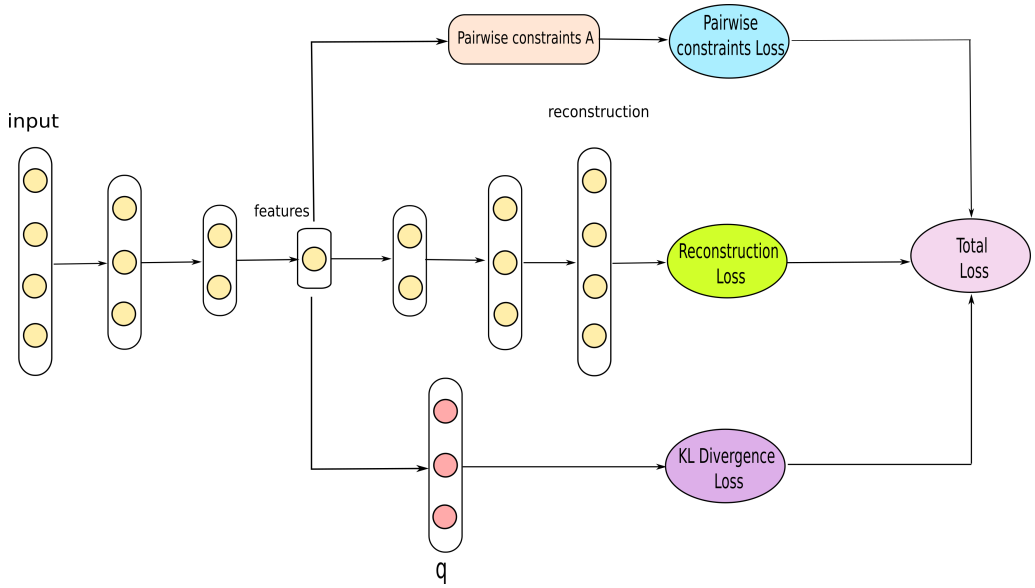


Figure 5.3: Our approach of Semi-Supervised Deep Embedded Clustering algorithm.

Pairwise constraints are determined only during the first epoch of training; the same

constraints are used in the next epochs as well. The derivative of loss with respect to the trainable parameters is backpropagated into the network, and $\frac{\partial L}{\partial \mu_j}$ is used to update the cluster centroids μ_j using gradient descent.

5.1.6 Evaluation

The important question is whether this semi-supervised retraining of the autonecoder is really beneficial to the clustering. In order to evaluate this, we used Dataset D3 (Section 1.5), but a different strategy compared to the previous clustering evaluation (see Section 4.1.6).

For each of the 4 recordings that comprise D3, we have annotated 1000 random pairs of vocalizations by again declaring whether they should belong to the same cluster (label 1) or not (label -1). These are the ground truth annotations. After a new clustering is produced, we can compare it to the ground truth annotations by checking the status of the annotated pairs in the produced clustering; by status, we mean whether they are assigned to the same cluster (label 1) or not (label -1). We can then compare the ground truth labels to the clustering-derived labels as in classification tasks, by filling a confusion matrix.

In general, random pairs used for ground truth annotations are most likely to have a -1 label, i.e. to belong to different clusters. This makes the classification task imbalanced, so the macro F1 metric is the most appropriate for the evaluation (see Section 2.8). We calculated the macro F1 score of the 4 recordings for Agglomerative clustering with 6 clusters 1) with hand-crafted features, 2) with deep features and 3) with deep features after retraining, setting a) 1, b) 3 and c) 5 pairwise constraints per batch and then their mean and standard deviation which are presented in Table 5.1. In case 3, since there is a randomness in the pairs we annotated during the training procedure, the resulting clustering differs from running to running, so we ran it 5 times per recording and calculated their mean, before calculating the mean for all recordings. The weights we chose were $\gamma_1 = 0.5$, $\gamma_2 = 0.2$ and $\gamma_3 = 0.001$. They were set after experimentation, though the best combination can vary depending on the recording and the autoencoder used. The retraining lasted up to 3 epochs, depending on the calculated loss after each epoch.

Clustering approach	Mean Macro F1 (+/- std) (%)
Simple	61.75 (+/- 2.85)
Deep	63.25 (+/- 2.59)
Deep retrained (1 pair/batch)	65.30 (+/- 2.02)
Deep retrained (3 pairs/batch)	65.00 (+/- 1.57)
Deep retrained (5 pairs/batch)	64.95 (+/- 2.61)

Table 5.1: Mean Macro F1 scores for the three clustering approaches.

It is obvious that the clustering resulting after retraining the model results in a better total score by approximately 2% compared to before retraining, also having a smaller standard deviation.

We can observe that by increasing the number of annotations per batch, there is a slight deterioration of the performance. However, the number of experiments is probably not big enough to allow us to draw an actual important conclusion about so small differences. In general, the more the annotations, the more robust the training is to individual confusing pairwise constraints (misannotated pairs); on the other hand too much intervention could destabilize the initial clustering patterns. In any case, we observed that the most suitable number of annotations varied depending on the recording, indicating that there is no "perfect number" of annotations. In general, though, 1 annotation per batch is much faster,

so it would make sense to try annotating more pairs if the user is not satisfied with this approach.

An important note is that the autoencoder used for the experiments presented above uses 4 output filters, instead of 8, for speed issues (explained in Section 5.2 below)

Of course, the metric employed provides only a general idea of the clustering performance, since we take into account only a very small subset of the total possible pairs of vocalizations. The best way to actually evaluate the clustering is by qualitative inspection of the clusters.

In any case, the proposed method describes an alternative to typical clustering, where the user can co-determine the results. It is possible that the user won't prefer the new clustering; however this functionality gives them the option of possible refinement of the already existing clustering. In general, the semi-supervised approach should be better seen as an alternative cluster proposition which may suit a specific application better and is more recording-specific.

5.2 Classification of online detected vocalizations

In the beginning of this chapter we mentioned that the ultimate goal would be to derive the best possible clustering to use it for classification purposes. This means that we will be able to record a mouse signal and provide the user with 1) detected vocalizations and 2) their class in online mode.

More specifically, after the clustering is produced, we can save each feature vector of the patterns (after the already described preprocessing) and its corresponding cluster label (target): $(\boldsymbol{x}_i, C_i)$. We then feed these pairs as training data to a classifier (Support Vector Machine with RBF kernel to be effective in non-linearly separable cases).

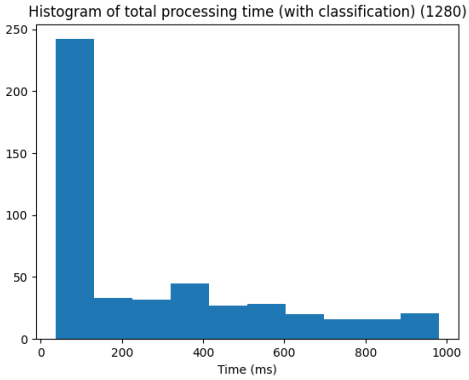
The next step is to use this model to classify newly emerging vocalizations to one of the defined classes simultaneously with detection, so that each category can be possibly connected with a specific mouse behavior. This means that each online detected vocalization must "pass" through the autoencoder model and then get preprocessed (again using Variance Thresholder, Standard Scaler and PCA, all defined by training data). This procedure has to be short enough in order to not impair the simultaneous recording.

Unfortunately, if we use the autoencoder with an encoder output dimension equal to 1280, the dimensionality reduction procedure of the outputs of the autoencoder is too slow for the online task, and as a result we had to retrain an autoencoder with output dimension equal to 640 (4 encoder output filters instead of 8). As stated in Figure 4.3 this choice deteriorates the features quality, though the clustering results are not much different. However, it significantly accelerates the dimensionality reduction process during the online task (see Figure 5.4a and 5.4b). As mentioned in Section 3.2, in online mode we process the recording using a 750 ms time frame, and since the recording is done simultaneously, we can't put up with a total processing time over 750 ms, which is unfortunately observed in Figure 5.4a. That's why we had to reduce the features size.

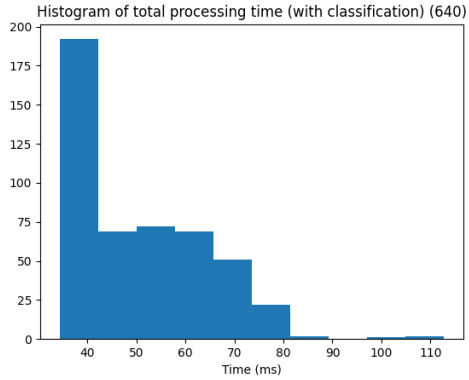
In Figures 5.5a and 5.5b, we also compare the time performance of the online mode with and without the classification functionality for a specific recording. It is obvious that the latency inserted with the classification is minor.

We have not evaluated the performance of the classifier on a test dataset, though training accuracy of the classifier is commonly around 95-97 %. In Figure 5.6 we present examples from actual clustering (training data) and classified vocalizations of an online task (test data).

It is obvious that there is a consistency between the results from training and test dataset, indicating a relatively successful generalization of the classifier.

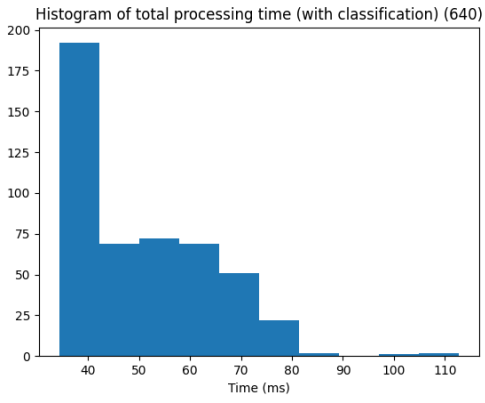


(a) Histogram of processing times with a 1280-dimensional encoder output.

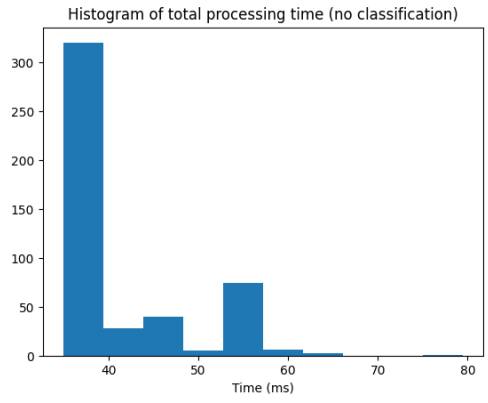


(b) Histogram of processing times with a 640-dimensional encoder output.

Figure 5.4: Histograms of total processing times of all 750 ms buffers that comprise the recording, for 1280- and 640-dimensional encoder outputs. In (a), we observe that processing time of several buffers surpassed 750 ms, which is unacceptable.



(a) Histogram of processing times with classification functionality.



(b) Histogram of processing times without classification functionality.

Figure 5.5: Histograms of total processing times of all 750 ms buffers that comprise the recording, with and without the classification functionality.

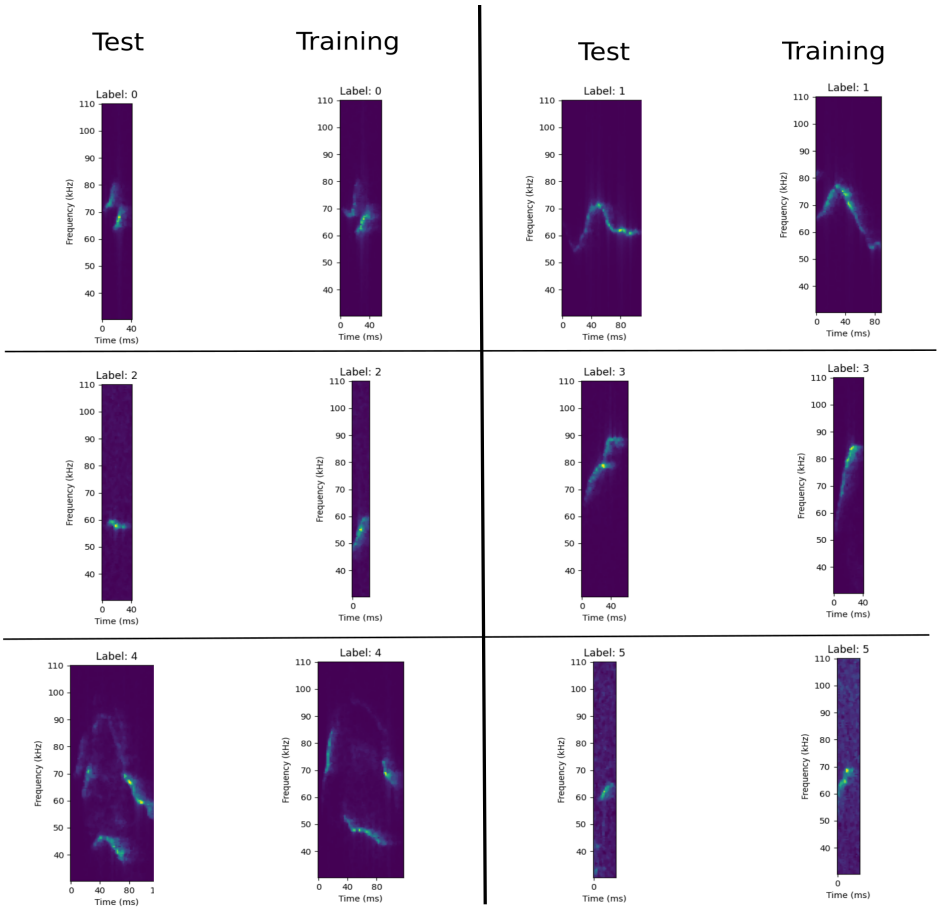


Figure 5.6: Examples of vocalizations of each class. On the right, the vocalizations are from the recording, whose clustering was used as training data for the classifier. On the left, the vocalizations are from a different recording, labeled automatically by the trained classifier.

Chapter 6

Conclusions and Future Work

In this chapter conclusions are summarized and presented, along with future thoughts, ideas and prospects for possible continuation and evolution of our work.

6.1 Conclusions

In this thesis we have presented Analysis of Mouse Vocal Communication (AMVOC), a tool for detecting, processing and analyzing ultrasound mouse vocalizations, which is characterized by novelties in many aspects. We would like to focus on the significant contributions of our work in USVs study.

- **Offline USVs detection**. Our approach has proved to perform very well in both clean and noisy backgrounds compared to other state-of-the-art tools, while it is also quite fast. The configurable nature of our methods enables the adjustments of the thresholding criterias to different recording conditions, thus guaranteeing a flexible and robust USVs detection tool.

- **Real-time USVs detection**. The ability of our detection method to be applied with minor adaptations in real time applications, with detection performance equivalent to offline approaches is quite revolutionary. Real-time feedback from mice vocal activity can be proven extremely useful for studying mice social behavior and also conduct experiments to test mice responses to specific stimuli.

- **Deep feature extraction and clustering**. We have developed a novel unsupervised approach for both *representing* and *grouping* USVs. Representations are derived from a deep convolutional autoencoder which we trained and multiple clustering algorithms can be used; resulting clusters can be thoroughly explored and evaluated by users. We have proposed a clustering evaluation procedure which involves human annotations and revealed that deeply extracted features outperform simple hand-crafted features (>30% in global- and cluster-level annotations), leading to possibly meaningful and homogeneous clusters, without limitations coming from predefined classes. While others have used autoencoders to analyze mouse USVs (Goffinet et al. 2021), these methods were not designed to allow users to explore the deep feature clustering and evaluate the results themselves.

- **Semi-supervised clustering approach**. We have presented a procedure for clustering refinement, which involves active learning. This means that the user can provide the model with information about whether certain USVs should be clustered together or not, thus providing a guiding line for vocalizations grouping. For the evaluation of this task we proposed a different approach, again involving some human

annotation and showing an improved performance of clustering methods compared to the completely unsupervised methodology. This functionality can be important for slightly differentiating clustering depending on the application and the user's needs.

- **Real-time USVs classification**. Another novelty is the opportunity of real-time classification of online detected USVs, using a specific clustering as training data for a classifier training. This further improves real-time applications, since the user will get feedback not only about when a mouse vocalized, but also what type of vocalization it produced, opening the door to associating behavior with syllables types and developing closed-loop behavioral assessments.

6.2 Future work and discussion

Our tool can be considered a complete USVs processing tool, though we hope that this work can inspire future directions that will give a helpful insight into mice communication and behavior understanding. These propositions and thoughts mainly focus on alternatives of the unsupervised and semi-supervised techniques we employed, and also suggest a broader exploration of USVs, either by studying whole sequences of syllables or by taking into account metadata from mice behavior during recording.

First of all, as far as the unsupervised feature extraction is concerned, we used a convolutional autoencoder. However, there are also other approaches for feature extraction, for example the use of a denoising autoencoder, which learns important features by trying to remove the noise from input images.

Another approach would be to use a generative model, e.g. Generative Adversarial Networks (GANs). These networks consist of a generator and a discriminator. Generators try to discover the data distribution and produce samples that mimic the inputs, whereas discriminators try to classify each sample as pure training data or data produced by the generator. The discriminator is in fact a classifier which has learned features from examples in the problem domain (Goodfellow et al. (2014)). This means that representations before the classification head can be useful feature vectors. The comparison between this approach and the autoencoder for the deep feature extraction can be very interesting.

In addition, the semi-supervised approach could be further studied, in the sense that it is interesting, and also expected, that the new clustering is heavily dependent on the pairs of syllables used in the active learning procedure. This makes sense, since they are the clustering guidelines used for training the model; in this context, the exploration of possible properties of vocalizations pairs that can ensure clustering performance enhancement is a promising future task.

In addition, different semi-supervised techniques for clustering improvement could be studied, which can explore other ways of human intervention in the training procedure. We have used the approach of imposing pairwise constraints to vocalizations; comparative analysis between our method and new ideas for semi-supervised learning techniques or procedures that insert the human factor in the loop would be interesting.

Another interesting subject would be an alternative clustering evaluation procedure; we have proposed two different methods which involve human annotations, though the possible discovery of some more automated and less subjective evaluation techniques could be beneficial and offer new insights into the clustering.

Another interesting aspect worth exploring is the possible temporal correlation between syllables, indicating standard patterns of occurring vocalizations. It has been known since mouse USVs were first identified as "songs" (Holy and Guo 2005) that across the timescale of a recording session mice will use similar sequences of vocalizations. There have been some attempts to describe the extent to which USV sequencing can be considered

patterned (Chabout et al. 2015, 2016), but these analyses have focused on pairwise changes in sequencing (i.e. how one syllable type follows or precedes another). We propose that future analyses would be best approached at varying timescales. By doing so, sequences of vocalizations could be processed and studied, so they can be connected and correlated with various behaviors in mice with a broader appreciation of vocal behavior.

This goal can also be achieved, if behavioral context is taken into account. This means that along with pure USVs information, data from experimental conditions during recording time could be used in machine learning algorithms. For example, the simultaneous use of metadata from the environment, the stimuli or the behavior of the mice could be integrated in the learning procedure and provide a context-aware USVs analysis. This approach is, in our opinion, of great interest, in both machine learning and neurobiology fields and can set the foundations for new prospects in addressing the social meaning of USVs.

Bibliography

Gustavo Arriaga, Eric P Zhou, and Erich D Jarvis. Of mice, birds, and men: the mouse ultrasonic song system has some features similar to humans and song-learning birds. *PloS one*, 7(10):e46610, 2012.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

J Michael Bowers, Miguel Perez-Pouchoulen, N Shalon Edwards, and Margaret M McCarthy. Foxp2 mediates sex differences in ultrasonic vocalization by rat pups and directs order of maternal retrieval. *Journal of Neuroscience*, 33(8):3276–3283, 2013.

Jack W Bradbury and Sandra Lee Vehrencamp. *Principles of animal communication*. Sinauer Associates, 2011.

Jonathan Chabout, Abhra Sarkar, David B Dunson, and Erich D Jarvis. Male mice song syntax depends on social contexts and influences female preferences. *Frontiers in behavioral neuroscience*, 9:76, 2015.

Jonathan Chabout, Abhra Sarkar, Sheel R Patel, Taylor Radden, David B Dunson, Simon E Fisher, and Erich D Jarvis. A foxp2 mutation implicated in human speech deficits alters sequencing of ultrasonic vocalizations in adult male mice. *Frontiers in behavioral neuroscience*, 10:197, 2016.

Jonathan Chabout, Joshua Jones-Macopson, and Erich D Jarvis. Eliciting and analyzing male mouse ultrasonic vocalization (usv) songs. *Journal of visualized experiments: JoVE*, (123), 2017.

Kevin R Coffey, Russell G Marx, and John F Neumaier. Deepsqueak: a deep learning-based system for detection and analysis of ultrasonic vocalizations. *Neuropsychopharmacology*, 44(5):859–868, 2019.

Francesca R. D'Amato, Elisabetta Scalera, Celeste Sarli, and Anna Moles. Pups call, mothers rush: Does maternal responsiveness affect the amount of ultrasonic vocalizations in mouse pups? *Behavior Genetics*, 36(3):471–471, may 2006. doi: 10.1007/s10519-006-9074-7. URL https://doi.org/10.1007%2Fs10519-006-9074-7.

Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, New York, 2 edition, 2001. ISBN 978-0-471-05669-0.

Allain-Thibeault Ferhat, Nicolas Torquet, Anne-Marie Sourd, Fabrice Chaumont, Jean-Christophe Olivo-Marin, Philippe Faure, Thomas Bourgeron, and Elodie Ey. Recording mouse ultrasonic vocalizations to evaluate social communication. *Journal of Visualized Experiments*, 2016, 06 2016. doi: 10.3791/53871.

Antonio HO Fonseca, Gustavo M Santana, Gabriela M Bosque Ortiz, Sérgio Bampi, and Marcelo O Dietrich. Analysis of ultrasonic vocalizations from mice using computer vision and machine learning. *eLife*, 10:e59161, mar 2021. ISSN 2050-084X. doi: 10.7554/eLife.59161. URL https://doi.org/10.7554/eLife.59161.

Jack Goffinet, Samuel Brudner, Richard Mooney, and John Pearson. Low-dimensional learned feature spaces quantify individual and group differences in vocal repertoires. *bioRxiv*, page 811661, 2021.

Rafael C Gonzalez and Richard E Woods. *Digital image processing*. Prentice Hall, Upper Saddle River, N.J., 3 edition, 2008. ISBN 9780131687288 013168728X 9780135052679 013505267X. URL http://www.amazon.de/Digital-Image-Processing-Rafael-Gonzalez/dp/013168728X/ref=sr_1_6?s=books-intl-de&ie=UTF8&qid=1330928076&sr=1-6.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Jasmine M. S. Grimsley, Jessica J. M. Monaghan, and Jeffrey J. Wenstrup. Development of social vocalizations in mice. *PLoS ONE*, 6(3):e17460, mar 2011a. doi: 10.1371/journal.pone.0017460. URL https://doi.org/10.1371%2Fjournal.pone.0017460.

Jasmine MS Grimsley, Jessica JM Monaghan, and Jeffrey J Wenstrup. Development of social vocalizations in mice. *PloS one*, 6(3):e17460, 2011b.

Frauke Hoffmann, Kerstin Musolf, and Dustin J. Penn. Spectrographic analyses reveal signals of individuality and kinship in the ultrasonic courtship vocalizations of wild house mice. *Physiology & Behavior*, 105(3):766–771, 2012. ISSN 0031-9384. doi: https://doi.org/10.1016/j.physbeh.2011.10.011. URL https://www.sciencedirect.com/science/article/pii/S0031938411004884.

Timothy E Holy and Zhongsheng Guo. Ultrasonic songs of male mice. *PLoS Biol*, 3(12):e386, 2005.

Luca Melotti, Sophie Siestrup, Maja Peng, Valerio Vitali, Daniel Dowling, Norbert Sachser, Sylvia Kaiser, and S Helene Richter. Individuality, as well as genotype, affects characteristics and temporal consistency of courtship songs in male mice. *bioRxiv*, 2021.

Joshua P Neunuebel, Adam L Taylor, Ben J Arthur, and SE Roian Egnor. Female mice ultrasonically interact with males during courtship displays. *eLife*, 4, may 2015. doi: 10.7554/elife.06203. URL https://doi.org/10.7554%2Felife.06203.

John Nyby, Gerard A. Dizinno, and Glayde Whitney. Social status and ultrasonic vocalizations of male mice. *Behavioral Biology*, 18(2):285–289, 1976. ISSN 0091-6773. doi: https://doi.org/10.1016/S0091-6773(76)92198-2. URL https://www.sciencedirect.com/science/article/pii/S0091677376921982.

Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall Press, USA, 3rd edition, 2009. ISBN 0131988425.

Yazhou Ren, Kangrong Hu, Xinyi Dai, Lili Pan, Steven Hoi, and Zenglin Xu. Semi-supervised deep embedded clustering. *Neurocomputing*, 325, 10 2018. doi: 10.1016/j.neucom.2018.10.016.

C. N. Slobodchikoff, William R. Briggs, Patricia A Dennis, and Anne-Marie C. Hodge. Size and shape information serve as labels in the alarm calls of Gunnison's prairie dogs Cynomys gunnisoni. *Current Zoology*, 58(5):741–748, 10 2012. ISSN 1674-5507. doi: 10.1093/czoolo/58.5.741. URL https://doi.org/10.1093/czoolo/58.5.741.

Elsa Steinfath, Adrian Palacios, Julian Rottschäfer, Deniz Yuezak, and Jan Clemens. Fast and accurate annotation of acoustic signals with deep neural networks. mar 2021. doi: 10.1101/2021.03.26.436927. URL https://doi.org/10.1101%2F2021.03.26.436927.

Jacqueline M Tabler, Maggie M Rigney, Gordon J Berman, Swetha Gopalakrishnan, Eglantine Heude, Hadeel Adel Al-Lami, Basil Z Yannakoudakis, Rebecca D Fitch, Christopher Carter, Steven Vokes, et al. Cilia-mediated hedgehog signaling controls form and function in the mammalian larynx. *Elife*, 6:e19153, 2017.

Ryosuke O. Tachibana, Kouta Kanno, Shota Okabe, Kohta I. Kobayasi, and Kazuo Okanoya. USVSEG: A robust method for segmentation of ultrasonic vocalizations in rodents. *PLOS ONE*, 15(2):e0228907, feb 2020. doi: 10.1371/journal.pone.0228907. URL https://doi.org/10.1371%2Fjournal.pone.0228907.

Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern recognition*. Academic Press, Burlington, MA; London, 2009. ISBN 9781597492720 1597492728 9780080949123 0080949126. URL http://www.books24x7.com/marc.asp?bookid=37213.

Maarten Van Segbroeck, Allison T Knoll, Pat Levitt, and Shrikanth Narayanan. Mupet—mouse ultrasonic profile extraction: a signal processing tool for rapid and unsupervised analysis of ultrasonic vocalizations. *Neuron*, 94(3):465–485, 2017.

Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 478–487, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/xieb16.html.